எளிய தமிழில் Deep Learning

[Machine Learning - பாகம் 2]

து.நித்யா

[illegible] [illegible]

Deep Learning

(Machine Learning - [illegible] 2)

[illegible]. [illegible]

nithyadurai87@gmail.com

[illegible] :
[illegible]
kaniyam.com

[illegible] - [illegible], gurulenin@gmail.com

[illegible] : [illegible]. [illegible] tshrinivasan@gmail.com

[illegible] [illegible] [illegible] 2020

[illegible] - Deep Learning – [illegible] Machine Learning [illegible]. [illegible], [illegible].

[illegible] “[illegible]” [illegible], 2012 [illegible]. [illegible] Deep Learning [illegible].

[illegible], [illegible] editor@kaniyam.com [illegible].

http://kaniyam.com/learn-deep-learning-in-tamil [illegible]. [illegible].

[illegible], [illegible].

[illegible].

[illegible].[illegible]

tshrinivasan@gmail.com

[illegible]
[illegible]

editor@kaniyam.com

□□□□□□□□□□□□

[illegible]

[illegible] .. CC-BY-SA [illegible], [illegible]

- [illegible].
- [illegible].
- [illegible].
- [illegible], [illegible], [illegible] www.kaniyam.com [illegible]. [illegible].

[illegible] :

http://static.kaniyam.com/ebooks/learn-deep-learning-in-tamil.odt

[illegible]:

“[illegible]

[illegible] "

[illegible], deep learning [illegible]. [illegible]! [illegible]. [illegible] deep learning -[illegible]. [illegible].

[illegible] Machine Learning [illegible]. [illegible]. [illegible].

http://kaniyam.com/learn-machine-learning-in-tamil [illegible]

[illegible]. [illegible]

[illegible]
17 [illegible] 2020

[illegible]: nithyadurai87@gmail.com

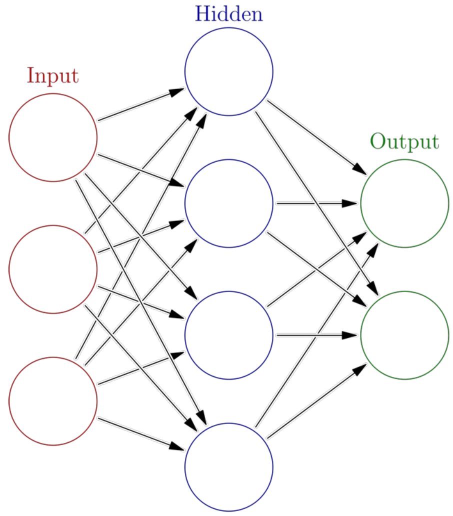
[illegible]: http://nithyashrinivasan.wordpress.com



1 Deep Learning

[illegible] (Machine Learning) [illegible]

[illegible] 3 [illegible] weights [illegible] bias-[illegible] weights [illegible] bias-[illegible] prediction-[illegible]



Deep Learning □□□□□□ □□□□□ □□□□□□□□ □□□□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□□□ □□□□□ □□□□□□□□□□□. Simple Neural Network, Shallow Neural Network, Deep Neural Network, Convolutional Neural Network, Recurrent Neural Network □□□□□□ □□□□□□□□ □□□□□□□□□□□ □□□□□□□ □□□□□□□□□□, □□□□□□□□ □□□□□ □□□□□□□□□□ □□□□□□□□ □□□□□□□□□□. □□□□□□□ □□□□□□□□□□□□□□□□□□ TensorFlow, PyTorch, Sonnet, Keras, Gluon, Swift, Chainer, DL4J, ONNX □□□□□□ □□□□□□□□ □□□□□□□□□□□□□□□□ deep learning-□□ □□□□□□. □□□□□□□□□ □□□□□□□ TensorFlow □□□□□□ deep learning-□□ □□□□□ □□□□□□□□ □□□□□□□□□□□ □□□□□ □□□□□□□□□□□□□ □□□□□□□□□. □□□□□□□□ □□□□□□□□ TensorFlow-□□ □□□□□ □□□□□□□□□□□□□ □□□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□□□□□□□□□□□.

2 TensorFlow

Deep Neural Network-□□ □□□□□□□□ □□□□□□□□□□□□□□ □□□□□□□□□□□□□ □□□□□□□ □□□□□□□□ 2011-□□ Google brain □□□□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□ Tensorflow □□□□□□. □□□ □□□□□□□□ □□□□□□□□ □□□□□ □□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□ □□□□□ □□□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□□ □□□ □□□□□ library □□□□□□. □□□□□ □□□□□□ □□□□□□□□ □□□□□□□ □□□□□□ □□□□□□□□ □□□□□ □□□□□□□□□□□ □□□□□□□□□.

□□□□□□□□ □□□□□ □□□□□□□□□□□ □□□□□□□□□ □□□□□□□□□□□□□□□□□□□□ □□□□□ data model-□□□□□ □□□□□□□□□ □□□□□□ □□□□□□□□□□□□□□□□□ (Data Model = Tensors). □□□□□□□□□ □□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□□□□□□ Tensorflow □□□□□□ □□□□□□□□□□□□□ (Data flow = Tensor flow). □□□□□□□□ □□□□□□□□□□□□□□□□□□ □□□ □□□□□□□ □□□□□□□□ Tensorboard □□□□□□ (Dashboard = Tensor board). □□□□□□□□□□□□□□ □□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□ □□□□□□□□□□□□□□□□□□□□ □□□□□□□ □□□ □□□□□□□□ □□□□□□ □□□□□□□□ □□□□□□□□□□□□□□□□□□□□. □□□□□□ Tensor flow graph □□□□□□

[illegible]. [illegible] nodes [illegible] edges [illegible] [illegible]. Nodes [illegible] edges [illegible] (Data flow graph = Tensor flow graph). [illegible] CPU, GPU, Server, Mobile [illegible].

[illegible]:

“hello world” [illegible]. [illegible] tensor flow [illegible]. [illegible] 3 [illegible].

i) [illegible] - Construction

ii) [illegible] - Running a session

iii) [illegible] - Routing & Analysis

https://gist.github.com/nithyadurai87/8c2d24e08cebd7dcff4696727c758913

```
import tensorflow as tf

a = tf.constant("hello world!")
print (a)

s = tf.Session()
print (s.run(a))
s.close()

with tf.Session() as s:
    print(s.run(a))
```

[illegible]:

1. [illegible] tensor flow library-[illegible] tf [illegible].

2. [illegible] library-[illegible] operators-[illegible] tf. [illegible]. [illegible] "hello world" [illegible] constant-[illegible]. [illegible] tf.constant() [illegible] a [illegible]. print (a) [illegible] constant [illegible]. [illegible]. [illegible].

3. [illegible]

[illegible]. [illegible] session() [illegible] operator [illegible]. [illegible]

[illegible]**1:** tf.session() [illegible] s [illegible]. [illegible] s.run() [illegible] constant-[illegible].. print (s.run(a)) [illegible] constant [illegible]. [illegible] session-[illegible] s.close(). [illegible].
[illegible]**2:** with tf.Session() as s: [illegible]. [illegible] session-[illegible].

[illegible]:

Tensor("Const:0", shape=(), dtype=string)

b'hello world!'

b'hello world!'

2.1 Constants

[illegible] tf.constant() [illegible] operator [illegible]. [illegible] string, int, float, bool [illegible].

https://gist.github.com/nithyadurai87/d7ede47ace7ce9028342f8153b81c9b3

```
import tensorflow as tf

print (tf.constant("hello world!"))
print (tf.constant(1))
print (tf.constant(1, tf.int16))
print (tf.constant(1.25))
print (tf.constant([1,2,3]))
print (tf.constant([[1,2,3],[4,5,6]]))
print (tf.constant([[[1,2,3],[4,5,6]],
                    [[7,8,9],[10,11,12]],
                    [[13,14,15],[16,17,18]]]))
print (tf.constant([True, True, False]))
print (tf.zeros(10))
```

```
print (tf.ones([10, 10]))
```

[illegible] [illegible] & [illegible]:

1. “ hello world” [illegible], [illegible] string [illegible] [illegible] [illegible]. dtype [illegible] data type-[illegible] [illegible].

Tensor("Const:0", shape=(), dtype=string)

2. [illegible] 1 [illegible], int32 [illegible]. [illegible] default [illegible]. [illegible] int16 [illegible], 1 [illegible], int16 [illegible].

Tensor("Const_1:0", shape=(), dtype=int32)

Tensor("Const_2:0", shape=(), dtype=int16)

3. [illegible] 1.25, float [illegible].

Tensor("Const_3:0", shape=(), dtype=float32)

4. [1,2,3] [illegible] list [illegible], [illegible] array-[illegible]. [illegible] shape [illegible], 3 [illegible]. [illegible] 1d array-[illegible] 3 columns [illegible].

Tensor("Const_4:0", shape=(3,), dtype=int32)

5. [[1,2,3],[4,5,6]] [illegible] array-[illegible], shape [illegible] (2,3) [illegible]. [illegible] 2d array-[illegible] 2 rows [illegible] row-[illegible] 3 columns [illegible].

Tensor("Const_5:0", shape=(2, 3), dtype=int32)

6. [[[1,2,3],[4,5,6]] ,

[[7,8,9],[10,11,12]] ,

[[13,14,15],[16,17,18]]] [illegible] array-[illegible], shape-[illegible] (3,2,3) [illegible]. [illegible] 3 [illegible] 2d array-[illegible]. [illegible] 2,3 [illegible] 2d array-[illegible] 2 rows, 3 columns [illegible].

Tensor("Const_6:0", shape=(3, 2, 3), dtype=int32)

7. True, False [illegible] bool [illegible].

Tensor("Const_7:0", shape=(3,), dtype=bool)

8. tf.zeros [illegible] tf.ones [illegible] 1-[illegible]. [illegible] 10 columns-[illegible] 1d array-[illegible], 10 rows & 10 columns -[illegible] 2d array-[illegible].

Tensor("zeros:0", shape=(10,), dtype=float32)

Tensor("ones:0", shape=(10, 10), dtype=float32)

2.2 Tensor properties

[illegible], [illegible], [illegible] 3 [illegible].

[illegible] **(Name):** [illegible], [illegible]. [illegible], [illegible] operator-[illegible]. [illegible], [illegible] operator-[illegible], [illegible] underscore (_) [illegible] 1,2,3, [illegible].

[illegible] **(Shape):** [illegible] shape=() [illegible]. [illegible], [illegible], [illegible]. [illegible], [illegible]. [illegible] d [illegible] 3 rows-[illegible] 1-[illegible] e [illegible]. tf.ones() [illegible] 1-[illegible].. [illegible] d.shape[0] [illegible] 3 rows-[illegible]. [illegible] columns-[illegible].

[illegible] **(data type):** int, float, string, bool [illegible] dtype [illegible]. [illegible] tf.cast() [illegible] operator [illegible]. [illegible] float [illegible] int-[illegible].

https://gist.github.com/nithyadurai87/70cafef972154224c891220aec2336c3

```
import tensorflow as tf

a = tf.constant(1)
b = tf.constant(2)
c = tf.constant(1, name = "value")
print (a.name)
print (b.name)
print (c.name)

d = tf.constant([[1,2],[3,4],[5,6]])
e = tf.ones(d.shape[0])
print (a.get_shape())
print (d.get_shape())
print (e.get_shape())

c = tf.constant(1.25)
print (c.dtype)
print (tf.cast(c, dtype=tf.int32).dtype)
```

[illegible]:

Const:0

Const_1:0

value:0

()

(3, 2)

(3,)

<dtype: 'float32'>

<dtype: 'int32'>

2.3 Operators

[illegible]

https://gist.github.com/nithyadurai87/811fb2f2d2871420f097c5c99c659f9b

```
import tensorflow as tf

a = tf.constant(121.5)
b = tf.constant([[1,2],[3,4],[5,6]])
c = tf.constant([[7,8],[9,10],[11,12]])

r1 = tf.sqrt(a)
r2 = tf.add(b,c)
r3 = tf.multiply(b,c)

print (s.run(r1))
print (s.run(r2))
print (s.run(r3))
```

[illegible]:

11.022704

[[8 10]

 [12 14]

 [16 18]]

[[7 16]

 [27 40]

 [55 72]]

2.4 Variables

[illegible] variable [illegible]. [illegible] weights, bias [illegible]. [illegible] variable-[illegible].

[illegible], a [illegible] variable [illegible]. [illegible] (v1) [illegible] [3,3] [illegible]. [illegible] 3 rows [illegible] 3 columns-[illegible].. [illegible] session-[illegible] a-[illegible], random-[illegible]. [illegible], tf.global_variables_initializer() -[illegible], "Attempting to use

uninitialized value v1" □□□□□ □□□□ □□□□□□□□□□□□□□ □□□□□□□. □□□□ □□□□ function-□ □□□□□□□□□□□□□□□□ □□□□□□□□□□□□□ variables-□□□□□□□□ □□□□ □□□□□□ □□□□□□□□□ initialize □□□□□□□□□□□□. □□□□ □□□ variable-□□□□□ □□□□□□□□□□□□□□□□□ □□□□□□□ □□□□ function-□ □□□□□□□□□ □□□ □□□ □□□□□□□□□.

□□□□□□□□□ b □□□□□ variable □□□□□□□□□□□□□□□□□□□□. □□□□ □□□□□□□□□□□□□□□□ initializer □□□□□ □□□□ variable-□□ □□□□ □□□□□□□□□□ □□□□□□ □□□□□□□□ □□ □□□□□□□□□□ □□□□□□□□ □□□□□□□□. □□□□□□ □□□□ □□□□□□□□□□ □□□□□□ □□□□□□□□ □□□□ □□□□□□□□□□ □□□□□□, □□□□ □□□□□□□ □□□□□□□□□□ □□□□□□□□ □□□□□□□□. □□□□ □□□□□□□□ □□□□□□ □□□□□□□□□□, □□□□ □□□□□□□ □□□□□□□ □□□□□□□□□□□□□□ □□□□□□□.

□□□□□□□□□ x □□□□□ variable 1 □□□□□ □□□□□□ □□□□□□□□□ □□□□□□□ □□□□□□□□□□□□□□□□□□□□. □□□□□□□□ session-□□□□□□ for loop □□□□□ 10 □□□□□□□□□ □□□□□□□□□, □□□□□□□ □□□□□□□□□□ □□□□ variable, 2 □□□□□ constant-□□□ □□□□□□□□□□□□□□□□□. □□□□□□□ □□□□□□□□□□□□□□ □□□□□□□ □□□□□□□, □□□□□□□ □□□□□□□□ □□□□□□□□□□□, x-□□ □□□□□□□□ tf.assign() □□□□□ □□□□□□□□□□□□□□□□. □□□□□□□□ □□□□□□□□ 1*2=2, 2*2=4 , 4*2=8, 8*2=16 … □□ □□□□□□□□□□.

https://gist.github.com/nithyadurai87/a2b16555bbe599d089969bf43318265e

```
import tensorflow as tf

a = tf.get_variable("v1", [3,3])
b = tf.get_variable("v2", initializer=tf.constant([[3, 3],[4, 4]]))

x = tf.Variable(1)
y = tf.constant(2)
r = tf.assign(x,tf.multiply(x,y))

with tf.Session() as s:
    #print(s.run(a))
    s.run(tf.global_variables_initializer())
    print(s.run(a))
    print(s.run(b))
    for i in range(10):
        print (s.run(r))
```

□□□□□□□□□□ □□□□□□□□:

[[0.71750665 -0.39080024 -0.01946092]

[0.94270015 0.61512136 -0.7205548]

[0.6800127 -0.81497526 0.7290914]]

[[3 3]
 [4 4]]
2
4
8
16
32
64
128
256
512
1024

2.5 Placeholders

Placeholders [illegible] session [illegible] run-time [illegible]. feed_dict [illegible] argument [illegible]. Variables [illegible]. [illegible] placeholders [illegible]. session [illegible]. [illegible], [illegible] x, y [illegible] 2 placeholders [illegible]. x-[illegible] y-[illegible]. [illegible] session [illegible] x-[illegible] 100, 200, 300 [illegible] random [illegible] 1 [illegible] 10 [illegible]. [illegible] y-[illegible].

https://gist.github.com/nithyadurai87/495e2484a602abc85d3e53ed7ace7407

```
import tensorflow as tf
import numpy as np

x = tf.placeholder(tf.float32,name="x")
y = tf.placeholder(tf.float32,[1],name="y")
```

```
z = tf.constant(2.0)
y = x * z

print (x)
with tf.Session() as s:
    print (s.run(y,feed_dict={x:[100]}))
    print (s.run(y,{x:[200]}))
    print (s.run(y,{x: np.random.rand(1, 10)}))
    print (s.run(tf.pow(x, 2),{x:[300]}))
```

Tensor("x:0", dtype=float32)

[200.]

[400.]

[[1.5783005 0.30819204 0.26068646 1.8491662 1.0529723 1.7923148
  0.9828862 1.5377688 0.06250755 1.2727137]]

[90000.]

[illegible]. [illegible] placeholders-[illegible]. [illegible] variables-[illegible]. [illegible] rows & columns [illegible]. [illegible] placeholders-[illegible]. Run-time [illegible].

2.6 Tensor board

Tensor board [illegible]. [illegible] x1, y1, c [illegible] 3 [illegible].

f = x1.y1 + squared(x1) + y1 + c

  = 5*6 + squared(5) + 6 + 5

  = 30 + 25 + 6 + 5

  = 66

[illegible] tf.summary.FileWriter() [illegible] class [illegible]. [illegible] 'tensorboard_example' [illegible]

directory-□□ □□□ folder-□ □□□□□□□□□□□□. □□□□□□□□□ □□□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□ (events & summaries) □□□□□□□□□ □□□□□□□□□. □□□□□ s.graph □□□□□ □□□□□□□□□ □□□□□□□□ □□□□□□□□□□□□□□.

https://gist.github.com/nithyadurai87/dbbe17d6036ea6188d9e581c3cbb2af6

```
import tensorflow as tf

x1 = tf.get_variable("a", dtype=tf.int32,  initializer=tf.constant([5]))
y1 = tf.get_variable("b", dtype=tf.int32,  initializer=tf.constant([6]))
c = tf.constant([5], name ="c")
f = tf.multiply(x1, y1) + tf.pow(x1, 2) + y1 + c

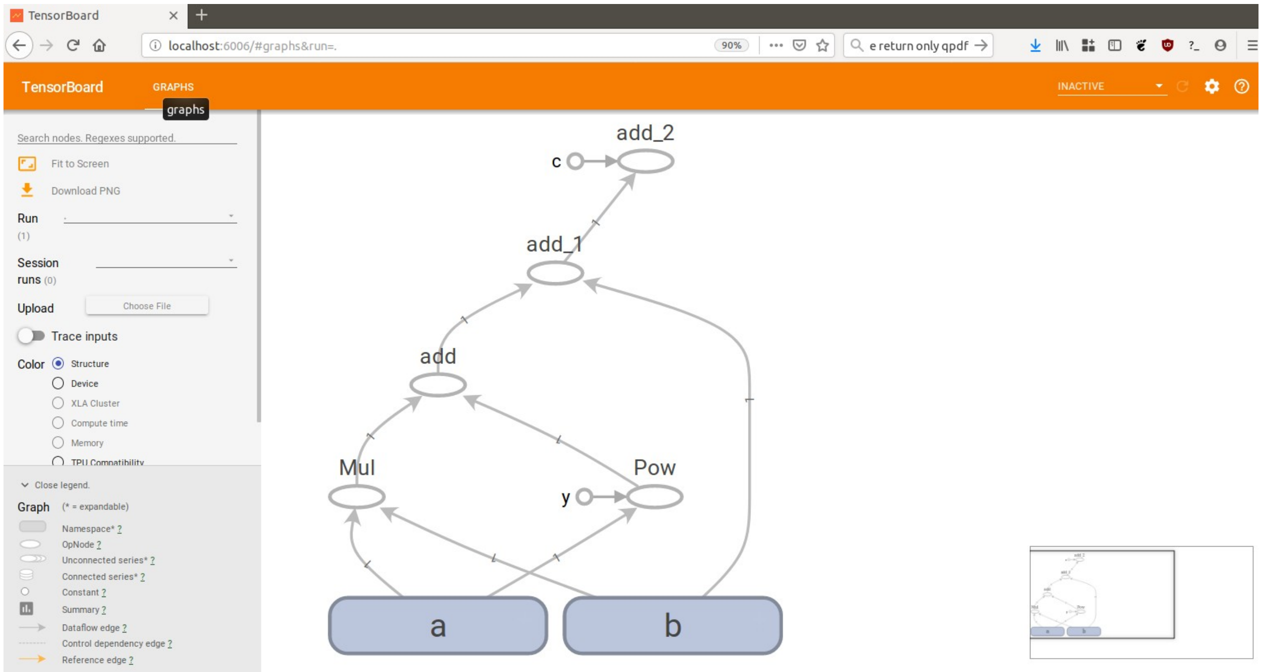
with tf.Session() as s:
    summary_writer = tf.summary.FileWriter('tensorboard_example',s.graph)
    s.run(tf.global_variables_initializer())
    print (s.run(f))
```

[66]

□□□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□ □□□□□□ □□□□□□□□□. □□□□□□□□ tensorboard-□ □□□□□□ □□□□□□□□□□ □□□□□□□□□ □□□□□□□□□□□□. □□□ 6006 port-□□ □□□□ □□□□□□□□□. □□□□ □□□□□□□□□ TensorBoard 1.13.1 at http://shrinivasan-Lenovo-Z50-70:6006 (Press CTRL+C to quit) □□□□□□□ □□□□□□□ □□□□□□□□□□□, tensorboard □□□□□□□□□□ □□□□□□□□□□□□□□□□□□ □□□□□□ □□□□□□□□□. □□□□ url-□□ □□□□□□□ □□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□□□.

```
$ tensorboard --logdir=tensorboard_example
```

□□□□□□□□□□□ □□□□□□□□ Tensor flow graph □□□□□□ □□□□□□□□□□□□□□□. □□□ nodes □□□□□□□□ edges □□□□□□ □□□□□□□ □□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□. add, mul, pow □□□□□□□□ nodes □□□□□□ □□□□□□□□□□□□□□□□. □□□ □□□□ □□□□□□□□□□□□□□□□□□ □□□□□□□□□□□. a,b,c □□□□□ □□□□□□ □□□□□□ variables, constants □□□□□□□ edges □□□□□□ □□□□□□□□□□□□□□□. □□□ □□□□□□□□□□□□ □□□□□□□□□□□□□.



3 PyTorch

Deep Neural Network-□□ □□□□□□□□□□□□ □□□□□□□□□□□ □□□□□□ □□□□□□□ □□□□□□□□ □□□□□□□□□□ PyTorch □□□□□. □□□ □□□□□□□□□ □□□□□□□ □□□□□□□□□□□□ □□□□□□□ □□□□ □□□□□ □□□□□□□□□□□□□□ □□□□□□ □□□□□□□□□□□□ □□□□□ □□□ library □□□□□. Torch □□□□□□□□□ □□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□□□□□ □□□□□□□□□□□□ □□□□□□□□ pytorch □□□□□.

3.1 Tensors

□□□□□□□ □□□□□□□□□□□□ □□□□□□□□□□ □□□□□□□ □□□□□□□□□ □□□□□□□□ □□□□□□□□□ □□□□□□□□□□ □□□□□□□□□ □□□□□□□□□□□□□□. □□□□□□□□□□□□□□ torch.Tensor() □□□□□ class □□□□□□ □□□□□□□ □□□□□□□□□ □□□□□□□□□□□□. □□□□□□□□□□ □□□□□□□□□□□ x1 □□□□□ □□□□□ □□□□□ □□□□ □□□□□□□□ □□□□□□□□□□□□□□□□□□□□. □□□□□□□ [1,2,3] □□□□□ 1d array-□□□□ □□□□□ x2 □□□□□□□□ □□□□□□□□□□□□□□□□□□□□□. □□□ 3 □□□□□□□□□□□□ □□□□□□□□□□□ □□□□□□□□□, □□□□ size □□□□□□□ 3 □□ □□□□□□□□□□□□□□ □□□□□□□. □□□□□□ □□□□□□□ □□□□□□□□□ □□□□□□□□□□□□, □□□□□□□□ □□□□□ □□□□□□□□□□□ Float32 □□□□□□□ □□□□□□□□□□□. □□□□□□□ □□□□□□□□ int64 □□□□□□□ □□□□□□□□ □□□□□□□□□□□, □□□□□ T-□□□□ □□□□□□ □□□□□ t -□ □□□□□ torch.tensor() □□□□□□ torch.as_tensor() □□□□ □□□□□□□□ □□□□□□ □□□□□□□ □□□□□□□□□□□ □□□□□□□□□□□□. □□□□□□□□□□□□□□

[illegible] x3, x4 [illegible]. [illegible] ([2, 4, 6]) [illegible]. [illegible] x2-[illegible] x3-[illegible].

[illegible] dtype, device, layout [illegible] 3 [illegible]. dtype [illegible]. device [illegible] CPU-[illegible] GPU-[illegible].layout [illegible] (memory layout) [illegible]. [illegible] torch.strided [illegible]. [illegible] torch.sparse_coo [illegible].

[illegible] eye(), zeros(), ones() [illegible], [illegible], 1-[illegible]. [illegible] rand() [illegible] 1d, 2d, 3d [illegible] random-[illegible].

https://gist.github.com/nithyadurai87/79d938591a3ddf6d2234dc54c441082c

```
import torch

x1 = torch.Tensor()
print (x1)

x2 = torch.Tensor([1,2,3])
print (x2)
print (x2.size())
print (x2.dtype)

x3 = torch.tensor([1,2,3])
x4 = torch.as_tensor([1,2,3])
print (x3)
print (x4)
print (x3+x4)
#print (x2+x3)

print (x4.dtype)
print (x4.device)
print (x4.layout)

print (torch.eye(2))
print (torch.zeros(2,2))
print (torch.ones(2,2))
print (torch.rand(2))
print (torch.rand(2,3))
```

```
print (torch.rand(2,3,4))
```

<u>□□□□□□□□□□□ □□□□□□□□□:</u>

tensor([])

tensor([1., 2., 3.])

torch.Size([3])

torch.float32

tensor([1, 2, 3])

tensor([1, 2, 3])

tensor([2, 4, 6])

torch.int64

cpu

torch.strided

tensor([[1., 0.], [0., 1.]])

tensor([[0., 0.], [0., 0.]])

tensor([[1., 1.], [1., 1.]])

tensor([0.7487, 0.0652])

tensor([[0.6830, 0.9479, 0.7002],[0.3283, 0.8602, 0.6711]])

tensor([[[0.8002, 0.9752, 0.4617, 0.5603],[0.1520, 0.6906, 0.9570, 0.7589],
[0.0122, 0.8932, 0.9644, 0.3375]],

[[0.5123, 0.3771, 0.5494, 0.1664],[0.0154, 0.4539, 0.8266, 0.8343],
[0.8994, 0.5009, 0.0348, 0.0757]]])

3.2 Some useful commands

topk() □□□□ □□□□□□□□□□□□ k=1 □□□□□□□□ □□□□□□□□□ □□□□□□ □□□□□□□□□□□□□, k=2 □□□□□□□□ □□□□□□ □□□□□□□ □□□□□□ □□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□. □□□□□□□□□□□□ □□□□□□□□□□□□□□ X □□□□□□ 2d □□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□. □□□□□□□□□ dim=1 □□□□□□□□□□ □□□□□□□□□□□□□□□□ 3 columns-□ 0,1,2 □□

[illegible], [illegible] row-[illegible] [illegible] 3 columns-[illegible] [illegible] [illegible] 120-[illegible], 2 [illegible] row-[illegible] [illegible] 3 columns-[illegible] [illegible] [illegible] 160-[illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] dim=0 [illegible] [illegible] 2 rows-[illegible] 0,1 [illegible] [illegible], [illegible] column-[illegible] [illegible] 2 rows-[illegible] [illegible] [illegible] 160-[illegible], 2 [illegible] column-[illegible] [illegible] 2 rows-[illegible] [illegible] [illegible] 120-[illegible], [illegible] column-[illegible] [illegible] 2 rows-[illegible] [illegible] [illegible] 90-[illegible] [illegible] [illegible] [illegible] [illegible].

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] torch.Tensor().item() [illegible] [illegible]. x.mul(2) [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] operation-[illegible] [illegible] underscore-[illegible] [illegible] (mul_()), [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].. [illegible] [illegible] underscore-[illegible] [illegible] [illegible] x-[illegible] [illegible] 2-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. reshape [illegible] [illegible] [illegible] [illegible] [illegible].

[illegible] [illegible] array-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] numpy() [illegible] [illegible]. [illegible] [illegible] array-[illegible] [illegible] [illegible] torch.from_numpy() [illegible] [illegible] [illegible] array()-[illegible] [illegible] [illegible].

[illegible] [illegible] [illegible](reshape) [illegible] view() [illegible] [illegible]. [illegible] [illegible] 2d [illegible] 1d-[illegible] [illegible], [illegible] [illegible] [illegible] [illegible] [illegible] view(1,6) [illegible] [illegible]. [illegible] [illegible] rows-[illegible] [illegible] [illegible] [illegible] columns [illegible] [illegible] row-[illegible] [illegible]. [illegible] view(1,-1) [illegible] [illegible]. -1 [illegible], [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible].

https://gist.github.com/nithyadurai87/8109fea71fd03258a494d4b8ee727a9a

```
import torch

x = torch.Tensor([[110,120,90],[160,20,60]])

print (x.topk(k = 1, dim = 1))
print (x.topk(k = 2, dim = 1))
print (x.topk(k = 1, dim = 0))
print (x.topk(k = 2, dim = 0))

print (torch.Tensor([110]).item())
print (x.mul_(2))

y = x.numpy()
z = torch.from_numpy(y)
```

```
print (type(y))
print (type(z))

a = x.view(1,-1)
print(a)
print(a.size())
```

[illegible] [illegible]:

torch.return_types.topk(values=tensor([[120.],[160.]]),indices=tensor([[1],
[0]]))

torch.return_types.topk(values=tensor([[120., 110.],[160.,
60.]]),indices=tensor([[1, 0],[0, 2]]))

torch.return_types.topk(values=tensor([[160., 120.,
90.]]),indices=tensor([[1, 0, 0]]))

torch.return_types.topk(values=tensor([[160., 120., 90.],[110., 20.,
60.]]),indices=tensor([[1, 0, 0],[0, 1, 1]]))

110.0

tensor([[220., 240., 120.],[40., 320., 180.]])

<class 'numpy.ndarray'>

<class 'torch.Tensor'>

tensor([[220., 240., 120., 40., 320., 180.]])

torch.Size([1, 6])

3.3 GPU

CPU [illegible] GPU [illegible]. [illegible] Graphics Processing Unit [illegible]. Graphics, 3d games [illegible], [illegible] GPU [illegible]. [illegible]!

[illegible] [illegible] random-[illegible] [illegible] [illegible] rows & columns-[illegible] [illegible] a [illegible] b [illegible] [illegible] 2d [illegible] [illegible]. [illegible] [illegible] [illegible] matmul() [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] start [illegible] end [illegible] [illegible] variables -[illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] CPU-[illegible] [illegible] [illegible] [illegible].

[illegible] [illegible] [illegible] GPU [illegible] [illegible] [illegible] cuda.is_available() [illegible] [illegible] [illegible]. [illegible] True [illegible] [illegible], [illegible] a, b [illegible] [illegible] [illegible] GPU-[illegible] [illegible] [illegible] [illegible] [illegible] .cuda() [illegible] function [illegible]. [illegible] [illegible] [illegible] GPU-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. CPU -[illegible] [illegible] [illegible] 22 [illegible] [illegible], GPU -[illegible] [illegible] [illegible] 0.0002 [illegible] [illegible] [illegible] [illegible] [illegible].

[illegible] [illegible] GPU [illegible], google colab [illegible] [illegible] (https://colab.research.google.com/notebooks/welcome.ipynb) [illegible] [illegible] [illegible] [illegible] [illegible].

https://gist.github.com/nithyadurai87/8eebce687ba439cd9b9691383c780dbe

```
import time
import torch

a = torch.rand(10000,10000)
b = torch.rand(10000,10000)
start = time.time()
a.matmul(b)
end = time.time()

print("{} seconds".format(end - start))

print (torch.cuda.is_available())

a = a.cuda()
b = b.cuda()

start = time.time()
a.matmul(b)
end = time.time()
print("{} seconds".format(end – start))
```

[illegible] [illegible]:

22.068870782852173 seconds

True

0.00020956993103027344 seconds

<class 'numpy.ndarray'>

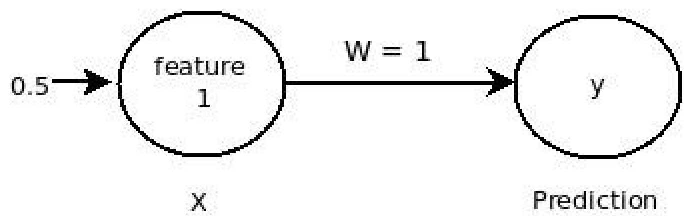
<class 'torch.Tensor'>

tensor([[220., 240., 120., 40., 320., 180.]])

torch.Size([1, 6])

4 Single Input Neuron

[illegible] [illegible] [illegible] [illegible] [illegible], [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] tensorflow [illegible] [illegible] [illegible] [illegible].



https://gist.github.com/nithyadurai87/4ad051ef5c3cc2a3da6c043cd99d0f1b

```
import tensorflow as tf

X = tf.constant(0.5)
Y = tf.constant(0.0)
W = tf.Variable(1.0)

predict_Y = tf.multiply(X,W)

cost = tf.pow(Y - predict_Y,2)
min_cost = tf.train.GradientDescentOptimizer(0.025).minimize(cost)

for i in [X,W,Y,predict_Y,cost]:
    tf.summary.scalar(i.op.name,i)

summaries = tf.summary.merge_all()

with tf.Session() as s:
    summary_writer = tf.summary.FileWriter('single_input_neuron',s.graph)
    s.run(tf.global_variables_initializer())
    for i in range(100):
        summary_writer.add_summary(s.run(summaries),i)
```

```
        s.run(min_cost)
```

[illegible] weight [illegible]. [illegible] 0.5 [illegible], [illegible] 1.0 [illegible] weight [illegible] predict_y [illegible]. [illegible] 0.0 [illegible] y [illegible]. [illegible] cost [illegible]. [illegible] square method -[illegible]. cost [illegible] gradient descent [illegible]. [illegible] learning_rate [illegible] parameter-[illegible] 100 [illegible] cost-[illegible].

[illegible]. [illegible], [illegible] list-[illegible] for loop-[illegible] scalar_summary() [illegible] function-[illegible]. [illegible] arguments-[illegible] list-[illegible], [illegible]. [illegible] (summary protocol buffer) [illegible]. [illegible] summary.merge_all() [illegible].

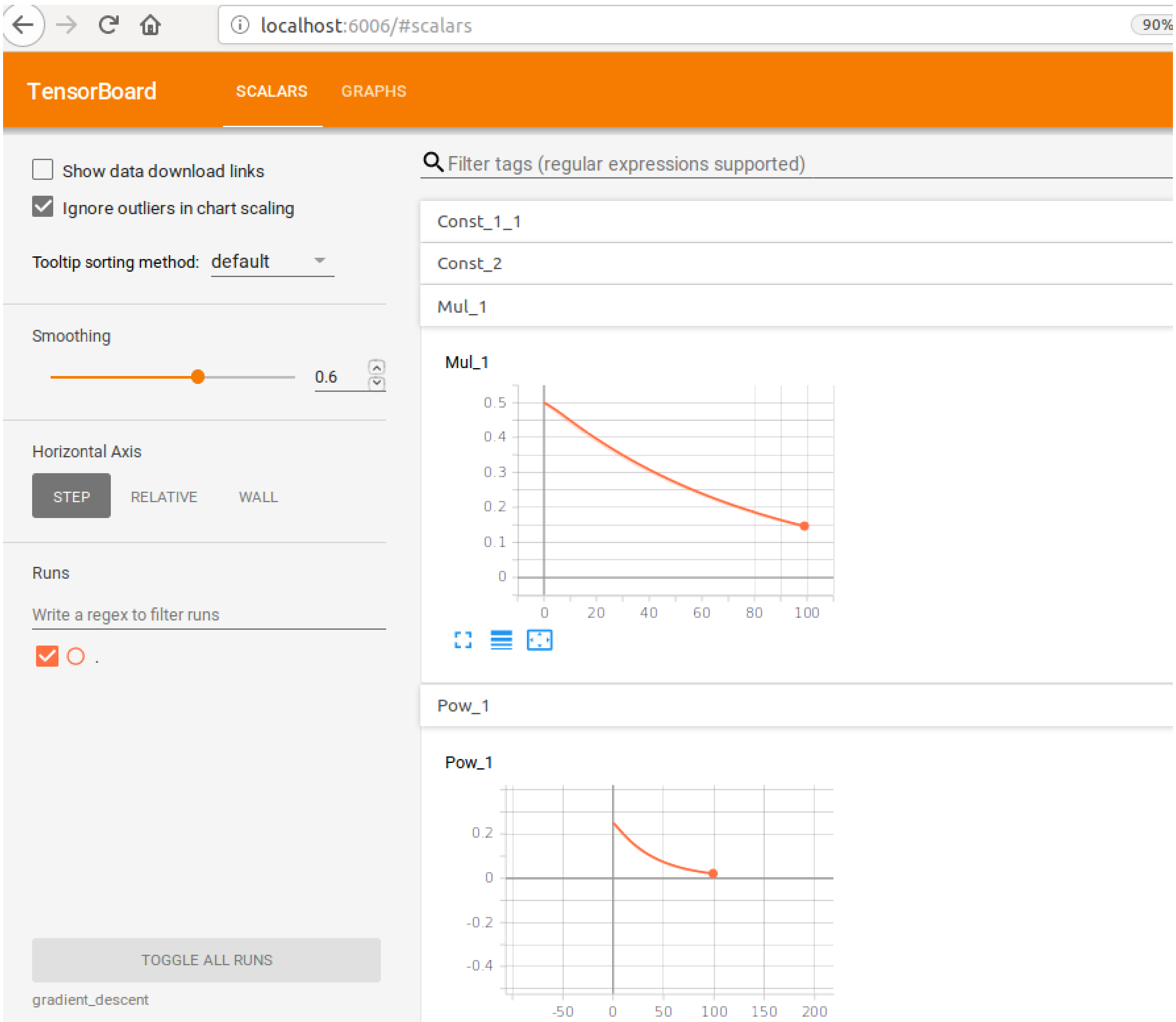
[illegible] session-[illegible] tf.summary.FileWriter() [illegible]. [illegible] directory-[illegible] 'single_input_neuron' [illegible] folder-[illegible]. [illegible] summary [illegible] asynchronous [illegible]. [illegible] for loop [illegible] 100 [illegible], [illegible] 0.025 [illegible] parameter [illegible] summaries-[illegible]. [illegible] (log files) [illegible].

[illegible].

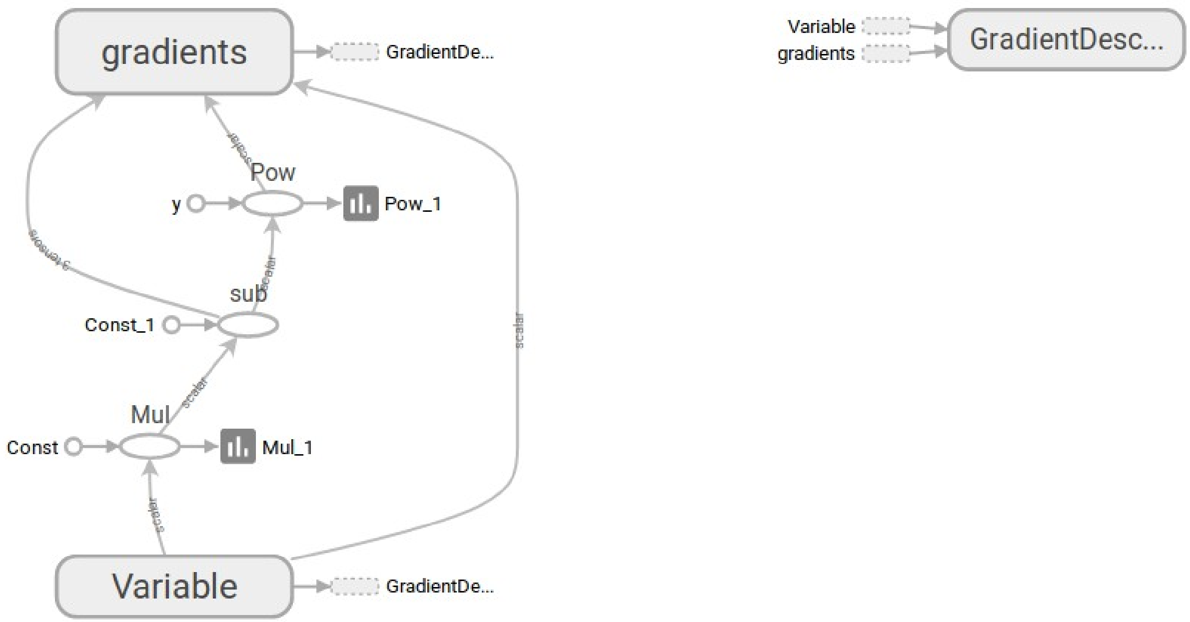
```
$ tensorboard --logdir=single_input_neuron
```

TensorBoard 1.13.1 at http://shrinivasan-Lenovo-Z50-70:6006 (Press CTRL+C to quit)

□□□□□□□□ Scalars □□□□□□□ Graphs □□□□□ □□□□□□ □□□□□□□□□ □□□□□□□□□□.
Scalar □□□□□ □□□□□□□□ list-□□□□□□ □□□□ □□□□□□□□ □□□□□□□□□□□□□□ □□□□□□□
□□□□□□□ □□□□□□□□□□□□□□□□□ □□□□□□□ □□□□□□□□□□□□□□ □□□□□□□□□□.
□□□□□□□□□□□□□□□□ predict_y □□□□□□□□□□□ □□□□□□□ 100 □□□□□□□□□□□□,
□□□□□□□ □□□□□□□□□□□□□ 0.0 □□□□□ □□□□□□□□ y-□□□□□□□□□□□ □□□□□□□□□□□□□
□□□□□□□ □□□□□□□□ □□□□□□ □□□□□□□□ □□□□□□ □□□□□□□ □□□□□□□□□□□□□□□□□.
□□□□□□□□□ Graphs □□□□□ □□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□ □□□□□□□□
□□□□□□□□□□□□□□□□ □□□□□□□ □□□□□□□ □□□□□□□□□□□□□□□□□.



4.1



5 Neural Networks

[illegible]

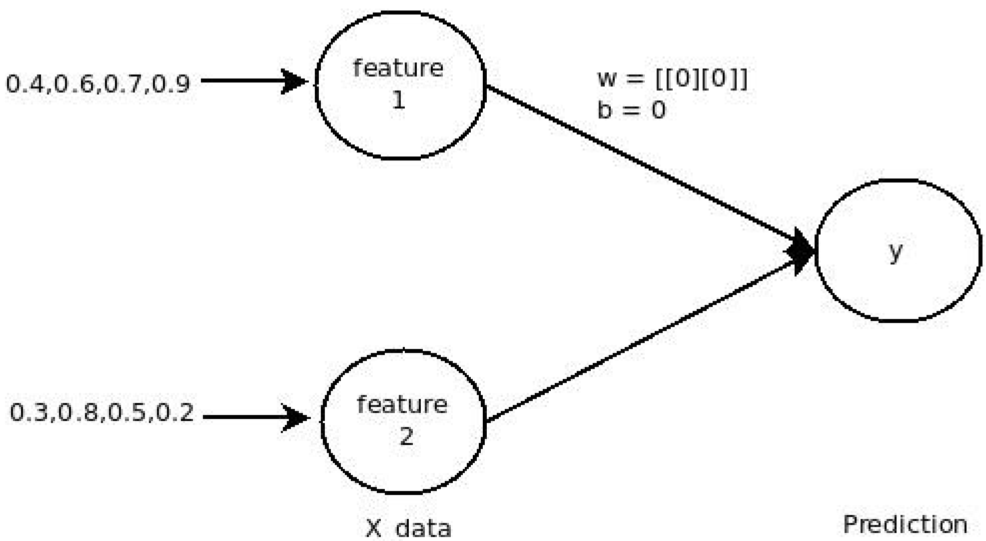
5.1 Python code

[illegible] (parameters – weights, bias) [illegible]

[illegible].

[illegible], [illegible], [illegible] "[illegible]" / Forward propagation [illegible]. [illegible] / Back propagation [illegible]. [illegible].

[illegible] 2d array-[illegible]. [illegible] 4 rows [illegible] row-[illegible] 2 columns [illegible]. [illegible] 2 features-[illegible] 4 [illegible]. [illegible] 2 features-[illegible] 2 [illegible]. [illegible] 0 [illegible] 1-[illegible]. [illegible] logistic regression-[illegible], [illegible] layer-[illegible] sigmoid activation fn [illegible].



https://gist.github.com/nithyadurai87/75e91a68e07d2e375aaf3ba4c87ef49a

```
from sklearn import datasets
import numpy as np

X_data = np.array([[0.4,0.3],[0.6,0.8],[0.7,0.5],[0.9,0.2]])
Y_data = np.array([[1],[1],[1],[0]])

X = X_data.T
Y = Y_data.T
```

```
W = np.zeros((X.shape[0], 1))
b = 0

num_samples = float(X.shape[1])
for i in range(1000):
    Z = np.dot(W.T,X) + b
    pred_y = 1/(1 + np.exp(-Z))
    if(i%100 == 0):
      print("cost after %d epoch:"%i)
      print (-1/num_samples *np.sum(Y*np.log(pred_y) + (1-Y)*(np.log(1-pred_y))))
    dW = (np.dot(X,(pred_y-Y).T))/num_samples
    db = np.sum(pred_y-Y)/num_samples
    W = W - (0.1 * dW)
    b = b - (0.1 * db)

print (W,b)
```

[illegible] [illegible]:

[illegible] [illegible] (Sample data):

[illegible] [illegible] [illegible] 4 [illegible] [illegible] (X_data), [illegible] [illegible] 4 [illegible] [illegible] (Y_data) [illegible] [illegible]. [illegible] [illegible] 0 & 1 [illegible] [illegible], [illegible] logistic regression-[illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] 4 rows & 2 columns [illegible] 2d array [illegible] [illegible]. [illegible] 4 sample data-[illegible] 2 features [illegible].

X_data = np.array([[0.4,0.3],[0.6,0.8],[0.7,0.5],[0.9,0.2]])

Y_data = np.array([[1],[1],[1],[0]])

[illegible] [illegible] (Input Layer):

[illegible]-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] features [illegible]. [illegible] features-[illegible] [illegible] [illegible] weights & bias [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] transpose [illegible] features*sample_records [illegible] [illegible] [illegible] [illegible] [illegible] (2,4). [illegible] [illegible] [illegible] [illegible] [illegible] shape-[illegible] (4,) [illegible] [illegible]. [illegible] transpose [illegible] (1,4)[illegible] [illegible] [illegible] [illegible]. [illegible] X, Y [illegible].

X = X_data.T

Y = Y_data.T

□□□□□□□□□□□□□ □□□□□□□□□□ □□□□□□□□□□ (Initializing weights & bias):

□□□□□□□ X.shape[0] □□□□□□ 2 features-□□□□□, X.shape[1] □□□□□□ □□□□□□□□□□□ □□□□□□□□□□□□□□□□ 4 □□□□□□□ □□□□□□□□□□□□ □□□□□□□□□□□□. weights-□ 0 □□ initialize □□□□□□□□□□ np.zeros() □□□□□□□□□□□□. □□□□ □□□□ □□□□□□□ □□□□□□□□□ □□□□ features-□□ □□□□□□□□□□□□□ □□□□□□ □□□□□□ □□□□□□□ □□□□□□□□□ X.shape[0] □□ □□□□□□□□□□□□□ □□□□□□□ □□□□ □□□□□□□□□□□□□□□□□□□. bias-□□□□ □□□□□□ □□□□□□□□ □□□□□□□□□ □□□□□□□□□□□□□□□.

```
W = np.zeros((X.shape[0], 1))

b = 0
```

□□□□□□□□□□ (Epochs):

X.shape[1] □□□□□□ □□□□□□□□□□□□ □□□□□□□□□□□□□□□□□ 4 □□□□□□□□ □□□□□□□□□ □□□□□□□□□□□□□□ □□□ □□□□□□ num_samples □□□□□ variable □□□□□□□□□□□□□□□□□□□. □□□ cost □□□□□□□□□□□□□□ □□□□□□□□□□□□□□. □□□□□□ □□□□□□□ □□□□□□□□□ □□□□□□□ □□□□□□□□□□□ □□□□□□□□□□ □□□□□ □□□□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□ □□□□□□□□□□□□□. □□□□□□□ □□□□□□□□□ □□ □□□□□□□□□□□□□□ □□□□□□□□□□□ □□□□□□□ □□□□□□□□□□ □□□□□□□□□ cost □□□□□□□□□□□□□□□□. □□□□ cost □□□□□□□ □□□□□□□□□ □□□□□□□□□□ (□□□□□□ □□□□ □□□□ □□□□□□□□□□ □□□□□ □□□□□□□□□□□□□□□□□) □□□□□□□□□□□□□ □□□□□□□□□□□□ □□□□□□ □□□□□□□□ □□□□□□□ □□□□□□□ cost □□□□□□□□□□□□□□□□. □□□□□□□ for loop □□□□□ 1000 □□□□□□□□□ □□□□□□□□□□. □□□□□□□ □□□□□□□□ 1 epoch / □□□□□□□□ □□□□□□□□□□□□□□□□□□□. □□□□□□□ epoch-□□□□ □□□□□□□□□□□□ □□□□ □□□□□□□ □□□□□□□□□□□ □□□□□□□□□ □□□□□□□□, □□□□□□□□□□□□□ □□□□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□□ □□□□□□□□ , □□□□□□ □□□□□□□ □□□□□□□□□ □□□□□□□□□□□ □□□□□□□□□□□□ □□□□□□□ □□□□□□□□□ □□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□ □□□□□□□□□□□ □□□□□□□□□□□□□□.

□□□□□□□□□□□□ □□□□□□□ (Forward Propagation):

□□□ □□□□□□□□ input features-□□ □□□□ □□□□□□□□□□□□□□ weights □□□□□□□ bias-□□□ □□□□□□□ □□□□ □□□□□□□□□□□□□ □□□□□□□□□ □□ □□□□□□□□ □□□□□□□□□□□. □□□□□□□ np.dot() □□□□□ □□□□□□□□□□□ □□□□□□□ □□□□□□□ □□□□□ □□□□□□ □□□□□□□□□ □□□□□ weights □□□□□□□ bias-□ □□□□□□□ Z -□□□ □□□□□□□□□□□□□. □□□□ Z □□□□□□ □□□ □□□□□□□□ □□□□□□□□□ linear □□□□□□□□ □□□□□□ □□□□□□□□□□ □□□□□. □□□□ logistic regression-□□□□ □□□□ □□□□□□□□ □□□□□□, Z □□□□□□□□ □□ sigmoid activation function-□□□□□□ □□□□□□□□□□□□□ 0 / 1 □□ □□□□□□□□□□□□□□□□. □□□ □□□□□□ □□□□□□□ □□□□□□□□, □□□□□□□□□□□ □□□□□□□□□□□ □□□□□□□ □□□□□□□□□□ □□□□□□□□□□□ □□□□□□ forward propagation □□□□□ □□□□□□□□□□□□□□□□□.

```
Z = np.dot(W.T,X) + b
pred_y = 1/(1 + np.exp(-Z))
```

[illegible] (Finding cost):

[illegible] (parameters = weights/bias) [illegible], [illegible]. [illegible], [illegible] cost [illegible]. [illegible] 1000 [illegible] print [illegible], [illegible] 100-[illegible] print [illegible] if-[illegible]. [illegible] 10 [illegible] cost [illegible].

```
if(i%100 == 0):
    print("cost after %d epoch:"%i)
    print (-1/num_samples *np.sum(Y*np.log(pred_y) + (1-Y)*(np.log(1-pred_y))))
```

[illegible] (Backward Propagation):

[illegible]. [illegible] delta / derivative [illegible]. derivative [illegible] ([illegible]) [illegible]. Gradient descent [illegible]. [illegible], [illegible]. [illegible] derivative [illegible]. [illegible].

```
dW = (np.dot(X,(pred_y-Y).T))/num_samples
db = np.sum(pred_y-Y)/num_samples
```

[illegible] (updating parameters):

[illegible] learning rate-[illegible] back propagation [illegible], [illegible]. learning rate [illegible] ([illegible] 0.1 [illegible]).

W = W - (0.1 * dW)
b = b - (0.1 * db)

[illegible](Finding right parameters):

[illegible]. [illegible].

weights = [[-3.44] [4.40]]

bias = 1.66

[illegible]:

cost after 0 epoch: 0.6931471805599453

cost after 100 epoch: 0.5020586179991661

cost after 200 epoch: 0.4448439151612328

cost after 300 epoch: 0.3979115275585397

cost after 400 epoch: 0.3590456846967788

cost after 500 epoch: 0.32654805177827606

cost after 600 epoch: 0.2990946818904699

cost after 700 epoch: 0.2756668906115973

cost after 800 epoch: 0.25548229634006936

cost after 900 epoch: 0.2379373586492477

[[-3.43906374] [4.4016814]] 1.6577403314904984

5.2 TensorFlow code

[illegible] TensorFlow [illegible]. [illegible]. [illegible] functions-[illegible] shallow neural network [illegible].

https://gist.github.com/nithyadurai87/fc1719d485dddfc4988c9e36975913c3

```
import tensorflow as tf

X_data = tf.constant([[0.4,0.3],[0.6,0.8],[0.7,0.5],
```

```
[0.9,0.2]],name="input_value")
Y_data = tf.constant([[1.0],[1.0],[1.0],[0.0]],name="output_value")

X = tf.transpose(X_data)
Y = tf.transpose(Y_data)

W = tf.Variable(initial_value=tf.zeros([1,X.shape[0]],
dtype=tf.float32),name="weight")
b = tf.Variable(initial_value=tf.zeros([1,1], dtype=tf.float32))

Z = tf.matmul(W,X) + b
cost = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=Z,labels=Y))
GD = tf.train.GradientDescentOptimizer(0.1).minimize(cost)

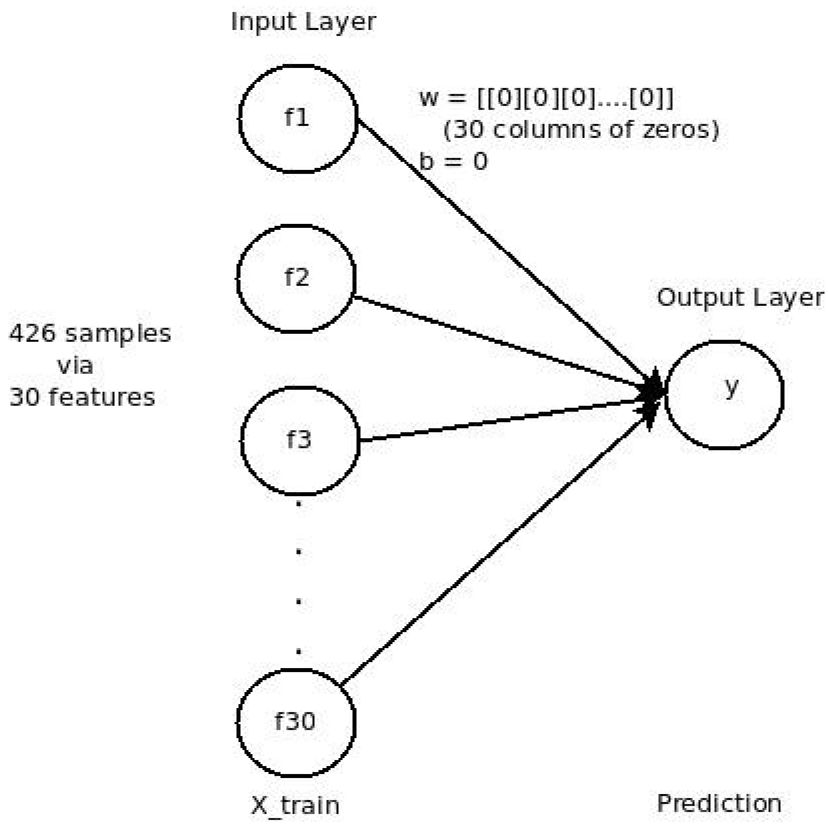
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(1000):
        c = sess.run([GD, cost])[1]
        if i % 100 == 0:
            print ("cost after %d epoch:"%i)
            print(c)
    print (sess.run(W),sess.run(b))
```

6 Simple Neural Networks

[illegible],, 1 [illegible], 2 [illegible]. [illegible] 30 features-[illegible] 426 [illegible]. [illegible] layer-[illegible]. [illegible] hidden layer-[illegible].

[illegible] sklearn-[illegible] datasets [illegible]. [illegible] 426 rows [illegible] 30 features-[illegible]. [illegible] train_test_split [illegible] normalize [illegible]. [illegible] 4000 [illegible], [illegible] 0.75 [illegible]. [illegible](X_train), [illegible](X_test) [illegible]. [illegible] 98% [illegible], [illegible]

[illegible] 93% [illegible].



https://gist.github.com/nithyadurai87/6c39697e3134a7ba5bd5cbe8790f95db

```
from sklearn import datasets
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

def normalize(data):
  col_max = np.max(data, axis = 0)
  col_min = np.min(data, axis = 0)
  return np.divide(data - col_min, col_max - col_min)

def model(X,Y):
  num_samples = float(X.shape[1])
  W = np.zeros((X.shape[0], 1))
  b = 0
  for i in range(4000):
    Z = np.dot(W.T,X) + b
    A = 1/(1 + np.exp(-Z))
```

```
    if(i%100 == 0):
      print("cost after %d epoch:"%i)
      print (-1/num_samples *np.sum(Y*np.log(A) + (1-Y)*(np.log(1-A))))
    dW = (np.dot(X,(A-Y).T))/num_samples
    db = np.sum(A-Y)/num_samples
    W = W - (0.75 * dW)
    b = b - (0.75 * db)
  return W, b

def predict(X,W,b):
    Z = np.dot(W.T,X) + b
    i = []
    for y in 1/(1 + np.exp(-Z[0])):
        if y > 0.5 :
            i.append(1)
        else:
            i.append(0)
    return (np.array(i).reshape(1,len(Z[0])))

(X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)
X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer,
random_state = 25)

X_train = normalize(X_train).T
y_train = y_train.T

X_test = normalize(X_test).T
y_test = y_test.T

Weights, bias = model(X_train, y_train)

y_predictions = predict(X_train,Weights,bias)
accuracy = 100 - np.mean(np.abs(y_predictions - y_train)) * 100
print ("Accuracy for training data: {} %".format(accuracy))

y_predictions = predict(X_test,Weights,bias)
accuracy = 100 - np.mean(np.abs(y_predictions - y_test)) * 100
print ("Accuracy for test data: {} %".format(accuracy))
```

[illegible] [illegible]:

cost after 0 epoch:

0.6931471805599453

cost after 100 epoch:

0.24382767353051085

cost after 200 epoch:

0.18414919195134818

……………………

cost after 3800 epoch:

0.06044063465393139

cost after 3900 epoch:

0.05993526502299061

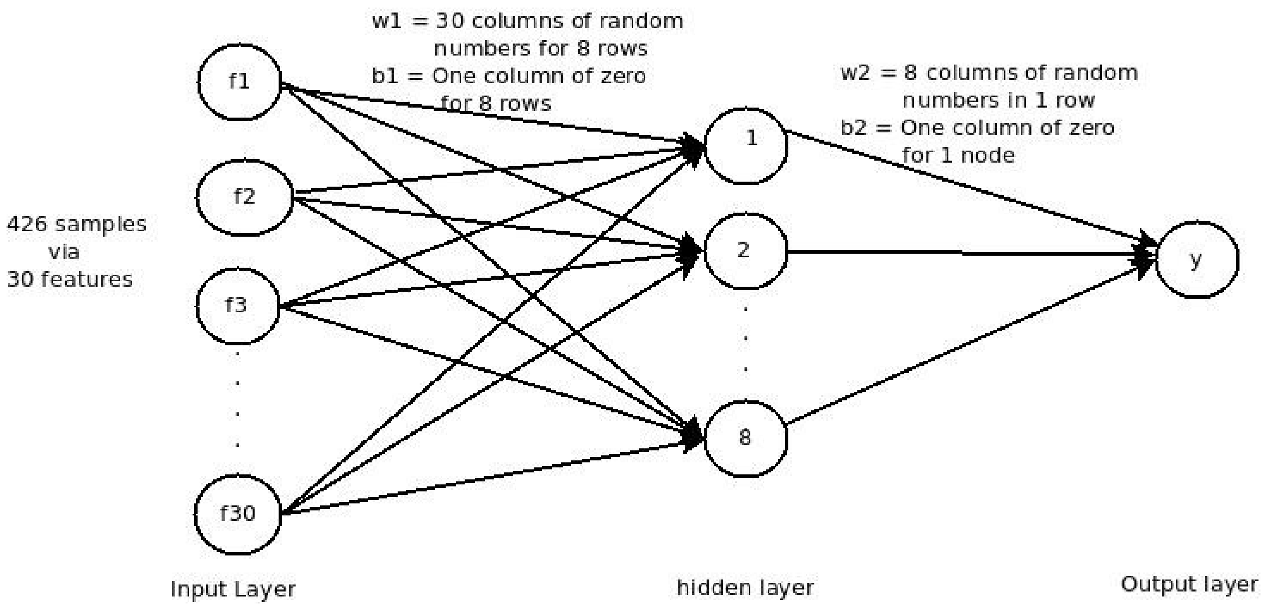
Accuracy for training data: 98.59154929577464 %

Accuracy for test data: 93.00699300699301 %

7 Shallow Neural Networks

Shallow [illegible] [illegible] [illegible] [illegible]. deep [illegible] [illegible] [illegible] [illegible]. [illegible] Deep [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] shallow [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] layer-[illegible] hidden layer [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] nodes-[illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] features [illegible] [illegible] [illegible] [illegible] [illegible].

[illegible] [illegible] [illegible] [illegible] train_test_split [illegible] [illegible] normalize [illegible] [illegible] , [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] 2 features-[illegible] [illegible] [illegible] (X_cancer[:,0], X_cancer[:,1]) [illegible] [illegible] [illegible] 0 [illegible] 1 [illegible] [illegible] (c=y_cancer) [illegible] [illegible] [illegible] [illegible] [illegible] [illegible], [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] logistic regression-[illegible] [illegible] [illegible] [illegible]. shallow neural network [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] tensor flow [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] tensor flow [illegible] [illegible] [illegible] [illegible].



https://gist.github.com/nithyadurai87/88a6f4e571200ce75de17464f9f13941

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
import tensorflow as tf
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

def normalize(data):
  col_max = np.max(data, axis = 0)
  col_min = np.min(data, axis = 0)
  return np.divide(data - col_min, col_max - col_min)

(X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)

cmap = matplotlib.colors.ListedColormap(['black','yellow'])
plt.figure()
plt.title('Non-linearly separable classes')
plt.scatter(X_cancer[:,0], X_cancer[:,1], c=y_cancer, marker= 'o', s=50,
cmap=cmap, alpha = 0.5 )
plt.show()
plt.savefig('fig1.png', bbox_inches='tight')

X_train, X_test, Y_train, Y_test = train_test_split(X_cancer, y_cancer,
random_state = 25)

X_train = normalize(X_train).T
```

```
Y_train = Y_train.reshape(1, len(Y_train))

X_test = normalize(X_test).T
Y_test = Y_test.reshape(1, len(Y_test))

num_features = X_train.shape[0]
X = tf.placeholder(dtype = tf.float64, shape = ([num_features,None]))
Y = tf.placeholder(dtype = tf.float64, shape = ([1,None]))

W1 = tf.Variable(initial_value=tf.random_normal([8,num_features], dtype =
tf.float64) * 0.01)
b1 = tf.Variable(initial_value=tf.zeros([8,1], dtype=tf.float64))
W2 = tf.Variable(initial_value=tf.random_normal([1,8], dtype=tf.float64) * 0.01)
b2 = tf.Variable(initial_value=tf.zeros([1,1], dtype=tf.float64))

Z1 = tf.matmul(W1,X) + b1
A1 = tf.nn.relu(Z1)
Z2 = tf.matmul(W2,A1) + b2

cost =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=Z2,labels=Y))
train_net = tf.train.GradientDescentOptimizer(0.2).minimize(cost)

init = tf.global_variables_initializer()
with tf.Session() as sess:
  sess.run(init)
  for i in range(5000):
    c = sess.run([train_net, cost], feed_dict={X: X_train, Y: Y_train})[1]
    if i % 1000 == 0:
      print ("cost after %d epoch:"%i)
      print(c)
  correct_prediction = tf.equal(tf.round(tf.sigmoid(Z2)), Y)
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
  print("Accuracy for training data:", accuracy.eval({X: X_train, Y: Y_train}))
  print("Accuracy for test data:", accuracy.eval({X: X_test, Y: Y_test}))
```

[illegible] [illegible]:

Input Layer-[illegible] placeholders-[illegible] [illegible]:

[illegible] [illegible] [illegible] training, testing [illegible] [illegible], normalize [illegible] reshape [illegible]. [illegible] , [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] X, Y [illegible] 2 placeholders [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] columns-[illegible] [illegible] [illegible] [illegible] [illegible]

features-[illegible] [illegible] [illegible] [illegible] [illegible] X_train.shape[0] [illegible], rows-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] None [illegible] X-[illegible] [illegible] [illegible]. [illegible] Y-[illegible] [illegible], [illegible] [illegible] target column-[illegible] [illegible] [illegible] records [illegible] [illegible] [illegible] [1,None] [illegible] [illegible].

```
X = tf.placeholder(dtype = tf.float64, shape = ([num_features,None]))
```

```
Y = tf.placeholder(dtype = tf.float64, shape = ([1,None]))
```

[illegible] [illegible] [illegible] [illegible] [illegible] X_train-[illegible] [illegible] 30 features-[illegible] placeholder nodes [illegible].

Hidden Layer-[illegible] [illegible] [illegible]:

placeholders [illegible] [illegible] feature-[illegible] [illegible] [illegible] [illegible] layer-[illegible] [illegible]. [illegible] layer [illegible] hidden layer. [illegible] shallow neural network [illegible] [illegible] [illegible] hidden layer-[illegible] [illegible]. [illegible] [illegible] nodes-[illegible] [illegible] [illegible] 8 [illegible] [illegible]. [illegible] [illegible] 8 nodes-[illegible] [illegible] (weights & bias) W1, b1 [illegible] [illegible] [illegible].

W1 [illegible] variable-[illegible] tf.random_normal() [illegible] 30 features-[illegible] [illegible] [illegible] node-[illegible] [illegible] random [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] hidden layer-[illegible] [illegible] [illegible] node-[illegible] random-[illegible] [illegible] 30 weights parameters-[illegible] [illegible]. [illegible] 30 columns-[illegible] 8 rows-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

b1 [illegible] variable-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. hidden layer-[illegible] [illegible] [illegible] node-[illegible] [illegible] [illegible] tf.zeros() [illegible] [illegible] column-[illegible] 8 [illegible] [illegible] [illegible] [illegible] [illegible].

```
W1 = tf.Variable(initial_value=tf.random_normal([8,num_features], dtype = tf.float64) * 0.01)
```

```
b1 = tf.Variable(initial_value=tf.zeros([8,1], dtype=tf.float64))
```

Output Layer-[illegible] [illegible] [illegible]:

[illegible] [illegible] [illegible] layer. [illegible] [illegible] W2, b2 [illegible] [illegible] [illegible]. [illegible] layer [illegible] [illegible] [illegible] node-[illegible] [illegible]. [illegible] [illegible] hidden layer-[illegible] [illegible] 8 nodes [illegible] [illegible] [illegible] features. [illegible] [illegible] 8 features-[illegible] [illegible] node-[illegible] [illegible] tf.random_normal([1,8]) [illegible] [illegible]. [illegible] [illegible] [illegible] node-[illegible] bias [illegible] [illegible] tf.zeros([1,1] [illegible] [illegible]. W2 =

```
tf.Variable(initial_value=tf.random_normal([1,8], dtype=tf.float64) * 0.01)
b2 = tf.Variable(initial_value=tf.zeros([1,1], dtype=tf.float64))
```

[illegible] [illegible] (Forward Propagation):

hidden layer-[illegible] [illegible] 8 [illegible] placeholders-[illegible] [illegible] [illegible] weights [illegible] bias-[illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] Z1 [illegible] [illegible]. [illegible] 0 [illegible] 1 [illegible] [illegible] [illegible] [illegible] sigmoid-[illegible] [illegible] relu [illegible] tanh [illegible] [illegible] [illegible]. hidden layer-[illegible] [illegible] 8 [illegible] [illegible] 8 [illegible] [illegible] [illegible]. [illegible] A1 [illegible]. [illegible] output layer-[illegible] [illegible] 1 [illegible] [illegible] weights [illegible] bias-[illegible] [illegible] Z2-[illegible] [illegible]. [illegible] [illegible] 0/1 -[illegible] [illegible] [illegible] [illegible]. Tensor flow-[illegible] [illegible] [illegible] layer-[illegible] [illegible] activate [illegible] [illegible] [illegible] [illegible] cost function-[illegible] [illegible] [illegible]. [illegible] [illegible] cost function [illegible] sigmoid [illegible] 0/1 [illegible] activate [illegible].

```
Z1 = tf.matmul(W1,X) + b1
A1 = tf.nn.relu(Z1)
Z2 = tf.matmul(W2,A1) + b2
```

[illegible] [illegible] (Finding cost):

sigmoid_cross_entropy() [illegible] Tensor flow-[illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] layer-[illegible] [illegible] activate [illegible] Z2 [illegible], [illegible] Y [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] tf.reduce_mean() [illegible].

```
cost =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=Z2,labels=Y))
```

Gradient descent [illegible] [illegible] [illegible]:

Tensor flow-[illegible] derivative-[illegible] [illegible], [illegible] [illegible] [illegible], [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. GradientDescentOptimizer() [illegible] function [illegible] [illegible] [illegible]. 0.2 [illegible] [illegible] [illegible] learning rate [illegible]. [illegible] [illegible] [illegible] cost [illegible] [illegible] [illegible] [illegible] minimize() [illegible] function-[illegible] [illegible].

```
train_net = tf.train.GradientDescentOptimizer(0.2).minimize(cost)
```

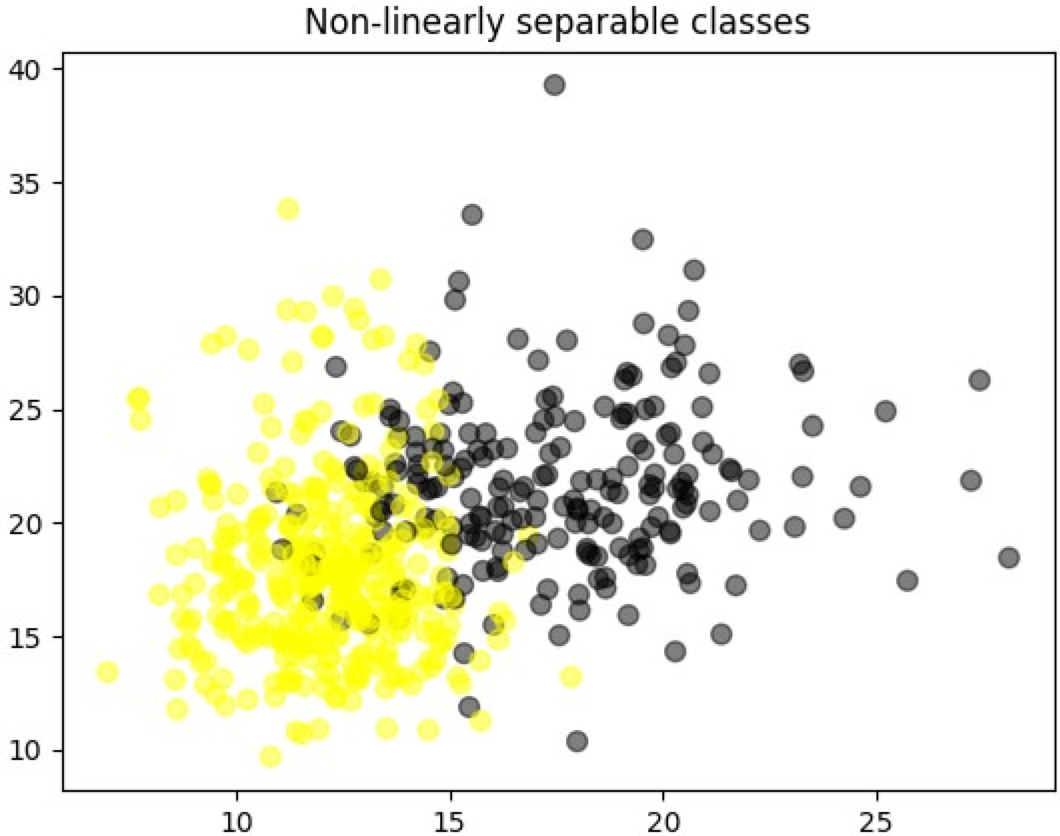
Session-[illegible] [illegible] [illegible] [illegible]:

tf.global_variables_initializer() [illegible] [illegible] [illegible] [illegible] variable [illegible] [illegible] [illegible], [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] session-[illegible] [illegible], [illegible] [illegible] [illegible] function-[illegible] [illegible] [illegible] [illegible]. [illegible] for loop [illegible] 5000 [illegible] [illegible], [illegible] [illegible] cost [illegible] [illegible], gradient descent [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].[illegible] X, Y -[illegible] placeholders-[illegible] [illegible] [illegible], X_train [illegible] Y_train -[illegible] [illegible] [illegible] feed_dict = {} [illegible] [illegible] [illegible]. [illegible] cost, 5000 [illegible] [illegible] [illegible] [illegible] gradient descent [illegible] [illegible]. [illegible] [illegible] c [illegible] variable-[illegible] [GD, cost] [illegible] [illegible] [illegible] [None,0.6931285163990998] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] cost-[illegible] [illegible] [illegible] sess.run()[1] [illegible] [illegible]. [illegible] 1000 [illegible] [illegible] cost-[illegible] [illegible] [illegible] if condition [illegible] [illegible].

[illegible] [illegible] [illegible] [illegible] [illegible]:

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] correct_prediction [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] accuracy [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] eval() function, [illegible] [illegible], [illegible] [illegible] palceholders-[illegible] [illegible] [illegible], [illegible] accuracy-[illegible] [illegible].

[illegible] [illegible]:



cost after 0 epoch:

0.6931285163990998

cost after 1000 epoch:

0.051629796747703585

…………………………

cost after 4000 epoch:

0.0307175769807556

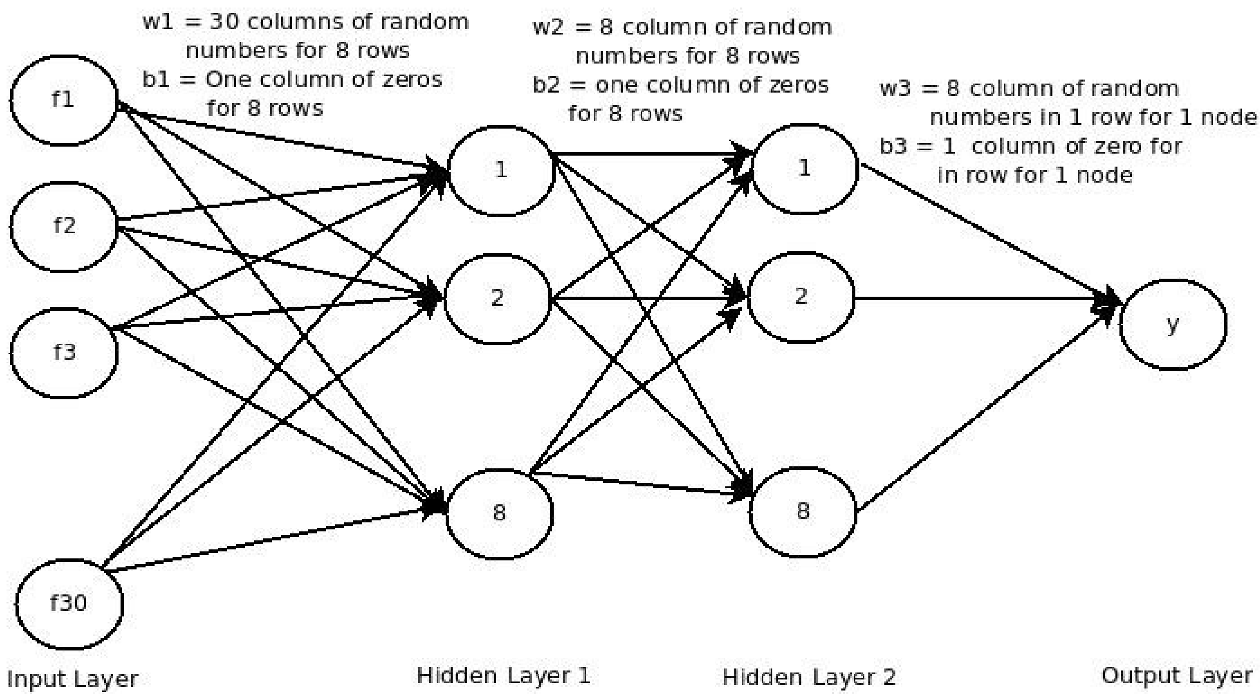
Accuracy for training data: 0.9906103

Accuracy for test data: 0.91608393

8 Deep Neural Networks

□□□□□□□□□□□□ □□□□□□□□□ hidden layers-□ □□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□□□ deep □□□□□□□□ □□□□□□□□□□□ □□□□□□□ multi-layer □□□□□□□□ □□□□□□□□□□□ □□□□□□

[illegible]. Shallow-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].. [illegible] [illegible] [illegible] [illegible] [illegible], [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].



https://gist.github.com/nithyadurai87/9d931f6b833bce58d6c2b0bdeedd266a

```
import numpy as np
import tensorflow as tf
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

def normalize(data):
  col_max = np.max(data, axis = 0)
  col_min = np.min(data, axis = 0)
  return np.divide(data - col_min, col_max - col_min)

(X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)

X_train, X_test, Y_train, Y_test = train_test_split(X_cancer, y_cancer, 
random_state = 25)
```

```
X_train = normalize(X_train).T
Y_train = Y_train.reshape(1, len(Y_train))

X_test = normalize(X_test).T
Y_test = Y_test.reshape(1, len(Y_test))

X = tf.placeholder(dtype = tf.float64, shape = ([X_train.shape[0],None]))
Y = tf.placeholder(dtype = tf.float64, shape = ([1,None]))

layer_dims = [X_train.shape[0],8,8,1]
parameters = {}
for i in range(1,len(layer_dims)):
  parameters['W' + str(i)] =
tf.Variable(initial_value=tf.random_normal([layer_dims[i], layer_dims[i-1]],
dtype=tf.float64)* 0.01)
  parameters['b' + str(i)] =
tf.Variable(initial_value=tf.zeros([layer_dims[i],1],dtype=tf.float64) * 0.01)

A = X
L = int(len(parameters)/2)
for i in range(1,L):
  A_prev = A
  Z = tf.add(tf.matmul(parameters['W' + str(i)], A_prev), parameters['b' +
str(i)])
  A = tf.nn.relu(Z)
Z_final = tf.add(tf.matmul(parameters['W' + str(L)], A), parameters['b' +
str(L)])

cost =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=Z_final,labels=Y))

GD = tf.train.GradientDescentOptimizer(0.1).minimize(cost)

with tf.Session() as sess:
  sess.run(tf.global_variables_initializer() )
  for i in range(5000):
    c = sess.run([GD, cost], feed_dict={X: X_train, Y: Y_train})[1]
    if i % 1000 == 0:
      print ("cost after %d epoch:"%i)
      print(c)
  correct_prediction = tf.equal(tf.round(tf.sigmoid(Z_final)), Y)
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
  print("Accuracy for training data:", accuracy.eval({X: X_train, Y: Y_train}))
  print("Accuracy for test data:", accuracy.eval({X: X_test, Y: Y_test}))
```

□□□□□□□□□□□ □□□□□□□□□:

[illegible] train_test_split [illegible], [illegible] normalize [illegible] reshape [illegible], [illegible], cost [illegible], gradient descent [illegible] cost [illegible], [illegible] accuracy [illegible] shallow [illegible] deep [illegible]. [illegible].

1. Shallow [illegible] deep [illegible], [illegible]. [illegible] hidden [illegible]. shallow-[illegible] hidden [illegible]. deep-[illegible]. [illegible] deep-[illegible] nodes-[illegible], [illegible]? [illegible] nodes [illegible]? [illegible]. [illegible], layer_dims = [X_train.shape[0],8,8,1] [illegible]. [illegible], [illegible] features-[illegible] X_train.shape[0] [illegible]. [illegible] 1 [illegible] nodes-[illegible]. [illegible] 8,8 [illegible] hidden [illegible], [illegible] 8 nodes [illegible].

```
layer_dims = [X_train.shape[0],8,8,1]
```

2. [illegible] Shallow-[illegible] hidden [illegible] 8 nodes-[illegible] w1, b1 [illegible], [illegible] 1 node-[illegible] w2, b2 [illegible]. [illegible] deep-[illegible] hidden [illegible] w1, b1 [illegible], [illegible] hidden [illegible] w2, b2 [illegible], [illegible] w3, b3 [illegible]. [illegible] 6 [illegible] for loop [illegible] parameters [illegible] dictionary [illegible]. [illegible] w1 [illegible]. [illegible] 30 features-[illegible] 8 rows-[illegible]. [illegible] w2 , w3 [illegible].[illegible] layer_dims[i], layer_dims[i-1] [illegible]. layer_dims[i] [illegible] weights [illegible] row-[illegible], layer_dims[i-1] [illegible] column-[illegible]. [illegible], for loop-[illegible] 1 [illegible] len(layer_dims) [illegible].

```
parameters = {}
```

```
for i in range(1,len(layer_dims)):
  parameters['W' + str(i)] = tf.Variable(initial_value=tf.random_normal([layer_dims[i], layer_dims[i-1]], dtype=tf.float64)* 0.01)
  parameters['b' + str(i)] = tf.Variable(initial_value=tf.zeros([layer_dims[i],1],dtype=tf.float64) * 0.01)
```

3. Shallow-□□ □□□□ □□□ hidden □□□□□ □□□□□□ □□□□□□□ □□□□□□□□□□□□□ □□□□□ features □□□□□□□□□□ weights □□□□□□□□□□ □□□□□□□□ Z1, A1 □□□□□ □□□□□□□□□□□ □□□□□□□□□□□. □□□□□□□□□ □□□□ □□□□□□□□□□ □□□□□ Z2 □□□□□ □□□□□□□□□ □□□□□□□ □□□□□□□□□□ activate □□□□□□□□□ cost-□□□□□□ □□□□□□□□□□ □□□□□ □□□□□□□□ □□□□□□□□□□□. deep □□□□□□□ □□□□□□□□□□ □□□□□ □□□□□□□□□ □□□□□ hidden □□□□□□□□□□ Z, A □□□□□□□□□□□□□□, □□□□□□□□□ hidden □□□□□□□□□□ Z, A □□□□□□□□□□□□□□, □□□□□ □□□□□□□□□□ □□□□□□□□□□ Z □□□□□□□□□□□□ □□□□□□□ □□□□□□□□. □□□□□□□ for loop □□□□□ □□□□□□□□□□□ □□□□□ □□□□□□□ A_prev □□□□□ variable-□□ X-□□ □□□□ features-□□ □□□□□□□□ □□□□□□□□□□□□ □□□□□ □□□□□□□□□□ Z , A □□□□□□□□□□ □□□□□□□□□□□□□□□□□□□. □□□□□□□ □□□□□ □□□□□ □□□□□□□□□ Z □□□□□□□ A_prev □□□□□□□□□ □□□□□□□□. □□□ features-□□ □□□□□□□ □□□□□□ □□□□□□□□□□ Z, A □□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□. □□□□□ □□□□□□□□□□ Z □□□□□□□ loop □□□□□□□ □□□□ □□□□□□□□□□□□□□□□□.

```
A = X
L = int(len(parameters)/2)
for i in range(1,L):
  A_prev = A
  Z = tf.add(tf.matmul(parameters['W' + str(i)], A_prev), parameters['b' + str(i)])
  A = tf.nn.relu(Z)
Z_final = tf.add(tf.matmul(parameters['W' + str(L)], A), parameters['b' + str(L)])
```

□□□□□□□□□□ □□□□□□□□:

cost after 0 epoch:

0.6931471575913913

cost after 1000 epoch:

0.6641999475918181

cost after 2000 epoch:

0.6641800975355493

cost after 3000 epoch:

0.2876489563036946

cost after 4000 epoch:

0.04967108208362405

Accuracy for training data: 0.9906103

Accuracy for test data: 0.91608393

9 Feed forward neural networks

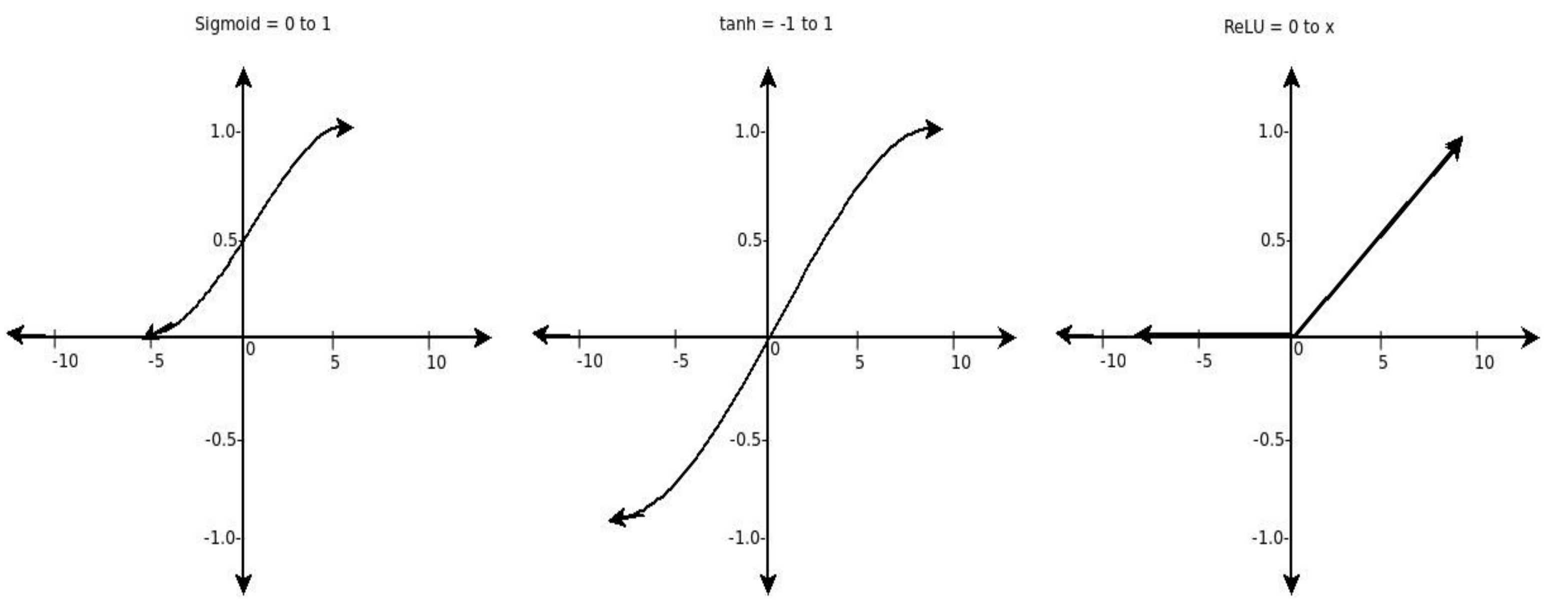
[illegible], [illegible] feed forward [illegible]. [illegible] deep layer-[illegible]. [illegible], [illegible] process [illegible]. [illegible]. [illegible] (feedback) [illegible]. [illegible], [illegible]. [illegible] feed forward neural network [illegible].

[illegible], [illegible] gradient [illegible] back propagation [illegible]. [illegible]. [illegible] ffnn [illegible]. [illegible] under-fitting [illegible], [illegible] over-fitting [illegible]. over-fitting [illegible] drop-out otimization [illegible]. [illegible]. [illegible] accuracy-[illegible]. [illegible] Regularization [illegible] optimization [illegible].

Different Classifiers:

feed forward □□□□□□□ □□□□□□□□□□□□ □□□□□□□□□ □□□□□□ □□□□□□□□□□□□□ ReLU □□□□□ □□□□□□□□□□□□□□□□, □□□□□ □□□□□□□□□ sigmoid □□□□□ □□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□. □□□□□□□ □□□□□□□ □□□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□? □□□□□□□□□□□ □□□□□□□ □□□□ □□□□□□□□ □□□□□□□□□□□□□ □□□□□ □□□□□□□.

- ➔ sigmoid - □□□ □□□□ □□□□□□□□□□□ 0 □□□□□ 1 □□□ □□□□□□□□□□. □□□□□□□□□□□ □□□□□□□□□□□ x-□□ □□□□□□□□□ □□□□□□□□ □□□□□□□□□□□□□ h(x), 0-□□□□□ 1-□□□ □□□□ □□□□□□□□□□□□ □□□□□□□ □□□□□□□□□□□ 1/(1+e**-x) □□□□□ □□□□□□□□□. □□□□□ sigmoid function-□□□□□ □□□□□□□□□ □□□□□□□□.
- ➔ Tanh - □□□ □□□□ □□□□□□□□□□□ -1 □□□□□ 1 □□□ □□□□□□□□□□. □□□□□□□ □□□□□□□□□□□□ □□□□□□□□□□□ [2/(1+e**-2x)]-1 □□□□□ □□□□□□□□□. □□□□□ □□□□□□□□□□ □□□□□□□ sigmoid □□□□□□ tanh □□□□□ □□□□□□□□ □□□□□□ □□□□□□□ □□□□□□□□□□□□□□.
- ➔ ReLU - □□□□ □□□□□□□ 0 □□□□□□ □□□□□□□□□□□□□ □□□□□□□. Rectified Linear Unit □□□□□□ Relu □□□□□□□□□□□□□□□□□□□. □□□□□ x-□□ □□□□□□□ 0-□□□□□ □□□□ □□□□□□□□□, □□□□ 0 □□□□□□, □□□□ □□□□□□□□□□ □□□□□□□□□□□□ □□□□□□□□□□□□□ □□□□□□□□□□. □□□□□□□□ □□□□ □□□□□□□□ max(0,x) □□ □□□□□□□□.

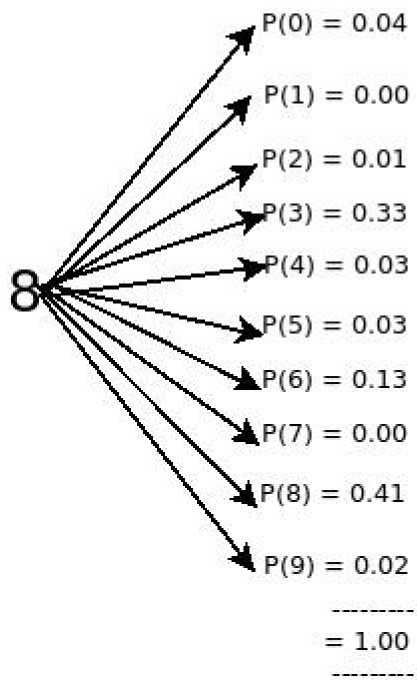


Machine learning-[illegible] [illegible] [illegible] [illegible], [illegible] sigmoid-[illegible] [illegible] logistic regression, neural networks [illegible] [illegible] [illegible]. [illegible] deep [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] back propagation [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] derivative [illegible] [illegible] [illegible] weights-[illegible] update [illegible]. [illegible] [illegible] sigmoid-[illegible] [illegible], [illegible] [illegible] 0 [illegible] 1 [illegible] [illegible] [illegible], x-[illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] hidden layer-[illegible] [illegible] 0 [illegible] [illegible] x-[illegible] [illegible] [illegible] ReLU-[illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] sigmoid [illegible].

10 Softmax neural networks

Softmax [illegible] multi-class classification-[illegible] [illegible] [illegible] [illegible] [illegible]. MNIST_data [illegible] [illegible] [illegible] [illegible] [illegible] 0 [illegible] 9 [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] 0 - 9 [illegible] 10 [illegible] label-[illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] multi-class classification-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] 55000 [illegible] [illegible] [illegible], 10000 [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

Softmax [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible], softmax() [illegible] [illegible] [illegible] 0-[illegible] [illegible] [illegible] [illegible], 1-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] 10 [illegible] class-[illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] 0 [illegible] 1 [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] 1 [illegible] [illegible]. [illegible] [illegible] [illegible] 8 [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible], [illegible] [illegible] [illegible] 1 [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].



https://gist.github.com/nithyadurai87/972397d9039d577a3be943779429f807

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
from random import randint

mnist = input_data.read_data_sets("MNIST_data/data", one_hot=True)
print (mnist.train.images.shape)
print (mnist.train.labels.shape)
print (mnist.test.images.shape)
print (mnist.test.labels.shape)

X = tf.placeholder(dtype = tf.float32, shape = (None,784), name='input')
Y = tf.placeholder(dtype = tf.float32, shape = ([None,10]))

W = tf.Variable(initial_value=tf.zeros([784, 10]), dtype = tf.float32)
b = tf.Variable(initial_value=tf.zeros([10], dtype=tf.float32))

XX = tf.reshape(X, [-1, 784])
Z = tf.nn.softmax(tf.matmul(XX, W) + b, name="output")

cost = -tf.reduce_mean(Y * tf.log(Z)) * 1000.0
```

```
GD = tf.train.GradientDescentOptimizer(0.005).minimize(cost)
correct_prediction = tf.equal(tf.argmax(Z, 1),tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction,tf.float32))

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    writer =
tf.summary.FileWriter('log_mnist_softmax',graph=tf.get_default_graph())
    for epoch in range(10):
        batch_count = int(mnist.train.num_examples/100)
        for i in range(batch_count):
            batch_x, batch_y = mnist.train.next_batch(100)
            c = sess.run([GD, cost], feed_dict={X: batch_x, Y: batch_y})[1]
        print ("Epoch: ", epoch)
    print ("Accuracy: ", accuracy.eval(feed_dict={X: mnist.test.images,Y:
mnist.test.labels}))
    print ("done")

    num = randint(1, mnist.test.images.shape[1])
    img = mnist.test.images[num]

    classification = sess.run(tf.argmax(Z, 1), feed_dict={X: [img]})
    print('Neural Network predicted', classification[0])
    print('Real label is:', np.argmax(mnist.test.labels[num]))
```

```
(55000, 784)
(55000, 10)
(10000, 784)
(10000, 10)
Epoch:  0
Epoch:  1
Epoch:  2
Epoch:  3
Epoch:  4
Epoch:  5
Epoch:  6
Epoch:  7
Epoch:  8
Epoch:  9
Accuracy:  0.9226
done
Neural Network predicted 3
Real label is: 3
```

11 Building Effective Neural Networks

[illegible] efficiency [illegible], Accuracy [illegible].

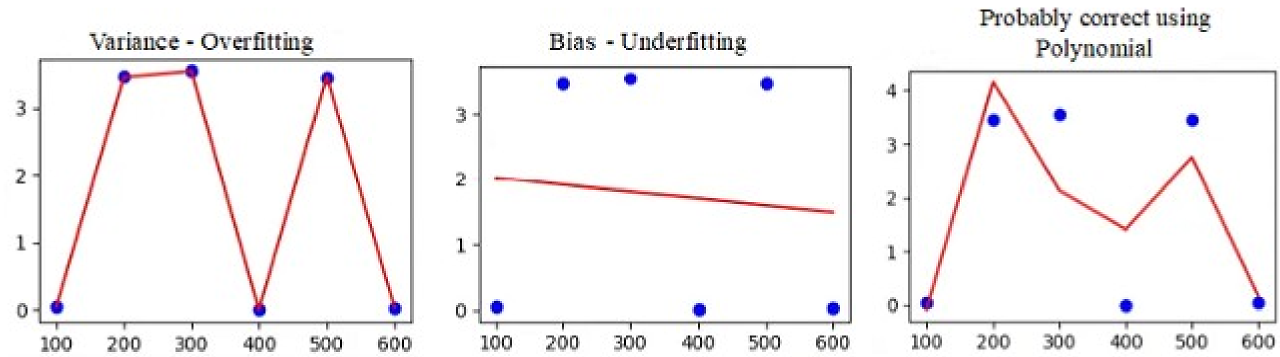
11.1 Bias-Variance Problem

[illegible] accuracy-[illegible], [illegible] bias [illegible] variance [illegible].

[illegible] features-[illegible] bias [illegible]. [illegible] features-[illegible]. [illegible], [illegible] features-[illegible] bias / under fitting [illegible].

[illegible] features-[illegible], variance / over fitting [illegible]. [illegible]. [illegible] features-[illegible], [illegible] features-[illegible].

[illegible]. [illegible] Machine Learning [illegible]. [illegible] 'Polynomial Regression’ [illegible].



□□□□□□□ bias-variance □□□□□□□□□□□□ □□□□□□□ □□□□□□□ □□□□ training, develop, testing □□ □□□□□□□□□ □□□□□□□□□□ □□□□□□□□□. □□□□ □□□□□ □□□□ □□□□□□□□□□□□□□ □□□□□ □□□□ □□□□□ □□□□□□□□□□□□ □□□□□□□□□□□□ □□□□□□□□□□ □□□□□□□□□□□□□□□□. training-□□□□□ □□□□□□□ □□□□□□ □□□□□□□ □□□□□□□ □□□□□ □□□□□□□□□□□□. develop-□□□□□ □□□□□□□ □□□□□□□□ □□□□□□ □□□□□□□□□□□□ □□□□□□ hyper-parameters □ □□□□□□□□□□□ □□□□□□ □□□□□□□□□ □□□□□□□□□. □□□□□□□ testing-□□□□□ □□□□□□□ □□□□□, □□□□□ □□□□□□□□□□□ □□□□□□□□□□□□□. □□□ □□□□□□□□□□□ □□□□□□□□□□□□□□□ cost □□□□□□□□□□ □□□□□□□□ □□□□□□ □□□□□ □□□□ □□□□□ □□□□□□□□□□□□ □□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□□□□□ □□□□□□□. □□□□□□□□□□□□□□□□□,

1. Training error 80% -□□□□□□ □□□□□□□ □□□□□□□□□ bias □□□□□□□□ □□□□□□□,

2. Training error □□□□□□□□□□□ (2%), Dev error □□□□□□□□□□□ (30%) □□□□□□□□□ variance □□□□□□□□ □□□□□□□,

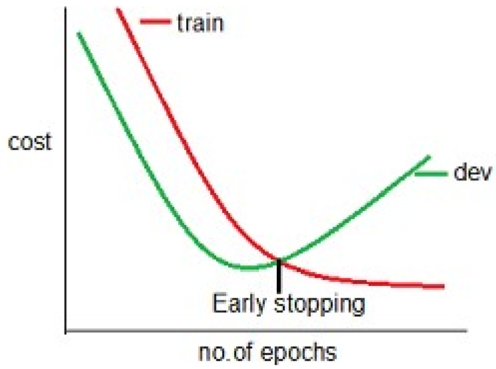
3. Training □□□□□□□ Dev □□□□□□□□□□ error □□□□□□□ □□□□□□□□□□□, Testing error □□□□□□□ □□□□□□□□□□□ □□□□□□□□□, variance □□□□□□□□ training & dev □□□□□□□□□□ □□□□□□□□□□□□□□ □□□□□ □□□□□□□□.

Early Stopping □□□□□□ □□□□□□□□□□□□□□□ □□□□□□□ □□□□□□ □□□ □□□□□□ □□□□□. Normalization □□□□□□ bias □□□□□□□□□□□□□□, Regularization □□□□□□ variance □□□□□□□□□□□□□□ □□□□□□□ □□□□□□□□□. weights decay, drop-out □□□□□□□ □□□□□□□ □□□□□□□□□□□□ □□□□□□□ □□□□□□□□□□□□□□□□□ □□□□□□□ □□□ regularization □□□□□□□□□□ □□□□□.

11.1.1 Early Stopping

Gradient Descent □□□□□ error □□□□□□□ □□□□□□ □□□□ □□□□□□□□□□□, Training □□□□□□□ Dev □□□□□□□□ □□□□□□□□□□□□□□□□□□ cost □□□□□□□ □□□ □□□□□□□□

[illegible]. Training [illegible] dev [illegible]. [illegible] dev [illegible], [illegible]. [illegible] 'Early Stopping' [illegible].



[illegible] (Convergence) [illegible].

11.1.2

11.1.3 Normalization

[illegible] feature column-[illegible], [illegible] -1 [illegible] +1 [illegible], [illegible] 0 [illegible] 1 [illegible] Normalization [illegible] feature scaling [illegible]. [illegible] mean normalization [illegible]. [illegible], [illegible] Machine Learning [illegible] 'Multiple Linear Regression' [illegible].

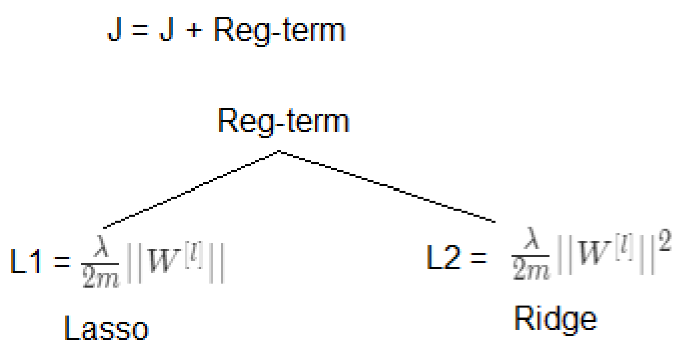
$$\frac{\text{particular value} - \text{mean of all values}}{\text{maximum} - \text{minimum}}$$

11.1.4 L1 , L2 Regularization

[illegible], [illegible]. [illegible]

[illegible] feature-[illegible] regularization-[illegible].

[illegible] L1 [illegible] L2 [illegible]. [illegible] features [illegible] L1 [illegible]. [illegible] features-[illegible] (0) [illegible]. [illegible] regression [illegible] L1-[illegible] Lasso Regression (Least Absolute shrinkage & selection operator) [illegible], L2-[illegible] Ridge regression [illegible]. L2 [illegible] /2m [illegible]. [illegible] L1 [illegible] L2 -[illegible] 'Penalty term' [illegible]. [illegible] cost (J) [illegible]. [illegible] L1 [illegible] 'Absolute value of Magnitude' [illegible], L2 [illegible] 'Squared Magnitude' [illegible].

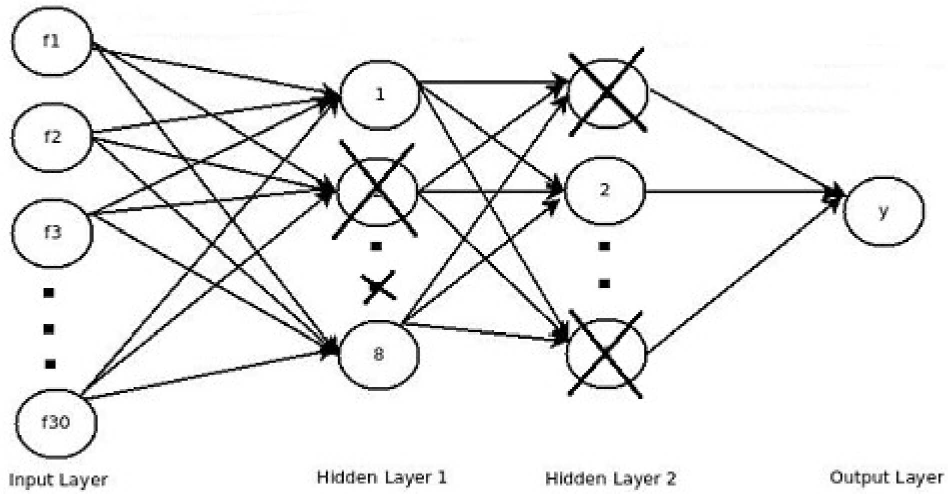


[illegible], overfitting [illegible]. [illegible] underfitting [illegible]. [illegible], [illegible].

11.1.5 Drop-out Regularization

[illegible] nodes-[illegible]

features-□□ □□□□□□□□□□□□□ □□□□□□□□□□□. keep_prob □□□□□□ □□□□ □□□□□□□□□□□□□□ parameter □□□□□. □□□□ □□□□□ □□□□□□□ □□□□□□□□□ □□□□□□□ nodes □□□□□□ □□□□□□□□ □□□□□□ □□□□□□□□□□□□□□□. □□□□□□ keep_prob = 0.6 □□□□□ □□□□□□□□□□ 60% nodes-□ □□□□□□□□ □□□□□□ □□□□ 40% nodes-□ □□□□□□ □□□□□□□ □□□□□ □□□□□□□□. □□□□□□□ □□□□□□□□□□□ □□□□□□□□ keep_prob □□□□□□□□□□ □□□□ □□□□□□□□□□□□□□.



11.2 Data-Insufficiency

□□□ □□□□□□□ □□□□□□□□□□□□□ □□□□□□□ □□□□□□□ □□□□□□□□□□□ □□□□□□□□□□□ □□□□□□□□ □□□□□. □□□□□□ □□□□□□□□ Data Augmentation □□□□ □□□□□□□□ □□□□□□□□□□□□□□. □□□□□□ □□□□□□□□□ □□□□□□□□□□□ □□□□□ □□□□□□. □□□□□□ □□□□□□□□ □□□□□□□□ □□□□□□□□□□ □□□□□□□□□ □□□□□□ □□□□ □□□□□□ □□□□□□□ □□□□□□□□□□□ □□□□□□□□□□□□□□□. Mirroring (□□□□□□□□□□□□□), Random cropping (□□□□□□ □□□□□□□□□□□□□□□□□), Rotation (□□□□□□□□□□), Shearing (□□□□□□□□□□), Colour shifting (□□□□□□□□□□ □□□□□□□□) □□□□□□□ □□□□□□□ □□□□□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□□□ □□□□□□□ □□□□□.

11.3 Vanishing & Exploding gradient

□□□□□□□ □□□□□□□□□□□□, □□□□ □□□□□□□□□□□□□□□□ weights-□□ □□□□□□□ □□□□□□□ □□□□□□□□ □□□□□□□□□, □□□□ gradients (dw, db) backpropagation-□□ □□□□ □□□ □□□□□□□□□□ □□□□□ □□□□□□□□□□□□□□. □□□□□ vanishing gradients □□□□□□□□□. □□□□□□ □□□□ □□□□□ gradients-□□ □□□□□□□ □□□□□ □□□□□□□□ □□□□□□ exploding gradients □□□□□□□□□. RNN □□□□□ □□□□□ □□□□□□□□□□

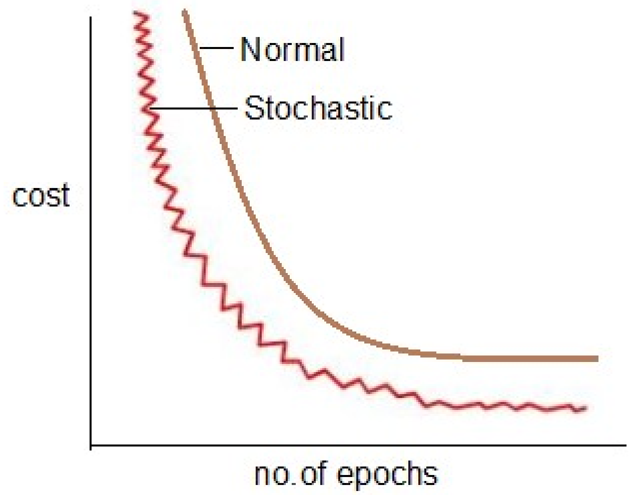
[illegible]. (RNN [illegible])

hidden layer-[illegible] tanh activation fn-[illegible]. [illegible] Xavier initialization [illegible] weights-[illegible], [illegible]. [illegible] weights matrix-[illegible], [illegible] variance = 2 / n (where n = no. of features)[illegible]. ‘Gradient Clipping’ [illegible] exploding [illegible]. [illegible] gradients-[illegible], [illegible] back propagation [illegible] weights [illegible].

11.4 Mini-batch Gradient descent

Gradient descent [illegible]. [illegible](convergence) [illegible], [illegible], [illegible]. [illegible] mini-batch gradient descent [illegible]. [illegible], [illegible] batch-[illegible] gradients [illegible]. [illegible], [illegible] 10 [illegible]. [illegible] gradients-[illegible]. [illegible] Stochastic Gradient descent [illegible]. [illegible], [illegible] Batch Gradient descent [illegible] Gradient descent without mini-batch [illegible].

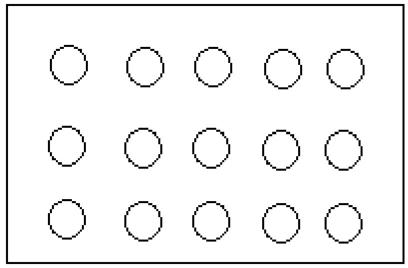
Stochastic [illegible] Batch [illegible] (Convergence using gradient descent)[illegible], Stochastic-[illegible]. [illegible] Batch-[illegible] mini batch-[illegible]. [illegible] Stochastic-[illegible].



11.5 Hyper-parameter Tuning

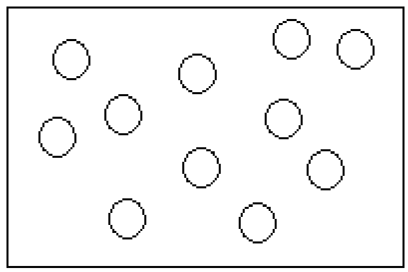
[illegible]. [illegible] weights , bias [illegible], [illegible]. [illegible]. [illegible], [illegible] nodes [illegible]. [illegible], [illegible] hyper-parameters [illegible]. Layers-[illegible] , Nodes-[illegible] , Mini batches-[illegible] , Gradient descent-[illegible] (learning rate) [illegible]. [illegible] hyper-parameter tuning [illegible]. [illegible]. [illegible].

11.5.1 Grid Search



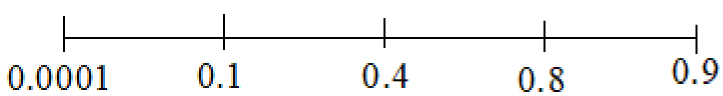
[illegible] combination-[illegible] Grid search [illegible].

11.5.2 Random Search



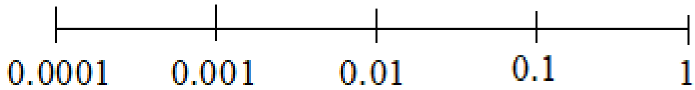
[illegible] Random Search [illegible], [illegible] grid search-[illegible].

11.5.3 Linear Scale



[illegible], Nodes-[illegible]. [illegible] non uniform-[illegible]. [Eg: 0.001, 0.1, …. 0.9]

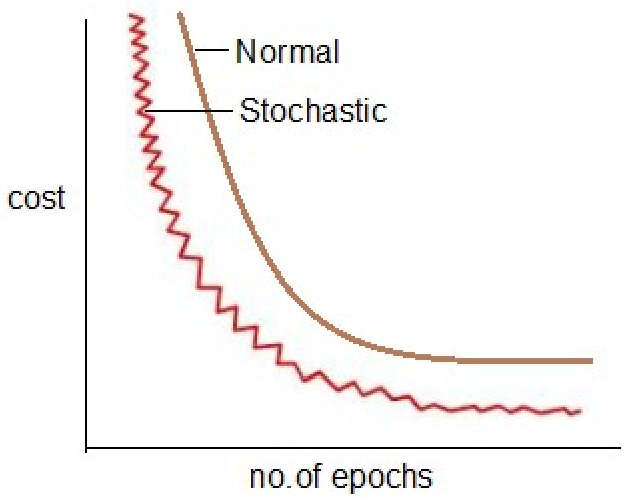
11.5.4 Log Scale



(learning rate) ... uniform- 0.0001, 0.001, 0.01, 0.1 ...

11.6 Optimization Techniques

hyperparameters- ... optimization ... Stochastic GD ...



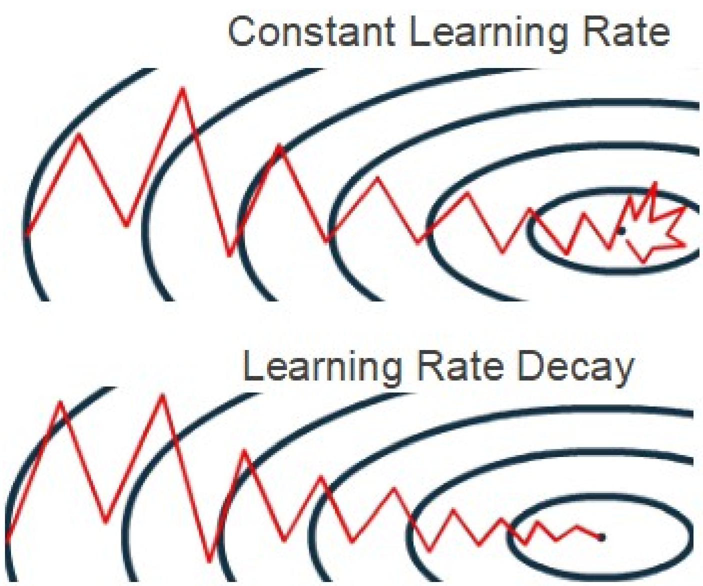
optimization ...

- Gradient Descent with Momentum ... (exponential moving average) ... oscillations- ...

□□□□□□□□□□□.

- Root Mean Square propagation □□□□□□□ RMS Prop □□□□□□□□□□□□□□□□□□□□□□. □□□□□□□ □□□□ □□□□□□□ □□□□□ □□□□□□□□□□□□□ □□□□□□□□□□□□□□ □□□□□ □□□□□□ □□□□□, oscillations-□□□ □□□□□□□□□□□□□□□ □□□□□□□ □□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□ RMS Prop □□□□□□□□□□□□□□□□□□□□□□.
- □□□□□□□□□□ □□□□□□□ □□□□□□□□□□□□□□ □□□□□□□ Adam optimization □□□□□□.

11.6.1 Learning Rate Decay



□□□ □□□□□□□□□□□□□ cost □□□□□□□□ gradient descent □□□□□□ □□□□□□□ □□□□□ □□□□□□□□□□□□, □□□□□ □□□□□□ □□□□□□□□□□□□□□□ □□□□□□□□□□□□ □□□□□□□□ □□□□□□□ □□□□□□ □□□□ □□□□□□□□□ □□□□□□□□□. □□□□□ □□□□□ □□□□□□□□□□□□□□ □□□□□□□□ □□□□□ cost □□□□□□□□ □□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□□ □□□□, □□□□□ □□□□□□□□□□□□□ □□□□□□ □□□□□□□□□□ □□□□□□□□□ □□□□□□ □□□□□□□ □□□□□□□ □□□□□□ □□□□□□□□ □□□□□□□□□□. □□□□□ □□□□□□□ □□□□□□□□□□ □□□□□ □□□□□□□ □□□□□□□□ □□□□□□□ □□□□□□□□□□□□ □□□□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□ □□□□□□□□□. □□□□□□□□ □□□□□□□ □□□□□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□□□□□□□ □□□□□□□□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□ learning rate decay □□□□□□. □□□ □□□□□□□□□□□□ □□□□□□ □□□□□□□□□ □□□□□□□□□ □□□□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□ □□□□□□□□□□□ □□□□□□□ □□□□□□. □□□□□ □□□□□□□ □□□□□□□□□□ □□□□□ □□□□□ □□□□□□□□□□□□□ □□□□□□□ □□□□□□□□□ □□□□□□□□□□ □□□□□□□□ □□□□□□□□□□ □□□□□□□□□□□□ □□□□□□□□□□ □□□□□□□□□□□□□□□□□. □□□□□□□ □□□□□□□□□□□□□□□

[illegible].

11.6.2 Pruning

[illegible]. [illegible]. Pruning [illegible]. [illegible].

11.6.3 Batch Normalization

Normalization [illegible] 'Multiple linear regression' [illegible]. [illegible], [illegible] 2 input features-[illegible] 450-sqft [illegible] 2500-sqft [illegible]. [illegible] 1 [illegible] 5 [illegible]. [illegible] input features-[illegible] -1 [illegible] +1 [illegible] Normalization [illegible].

[illegible] regression-[illegible] Normalization-[illegible] 'Batch Normalization' [illegible]. [illegible] mini-batch gradient descent [illegible]. [illegible]. [illegible] 'co-variance shift' (Change in data distribution of the input features) [illegible]. [illegible]. [illegible]. [illegible] normalize [illegible] 'Batch Normalization' [illegible]. [illegible]. CNN [illegible]-[illegible], [illegible] 'covariance shift' [illegible]. [illegible]. (CNN [illegible]

[illegible])

11.7 Example Program

deep neural network-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

1. [illegible] [illegible] train_test_split [illegible] [illegible] [illegible], X_train-[illegible] [illegible] [illegible] 2 features [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] non-linear [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

2. hidden [illegible] [illegible] [illegible] node-[illegible] [illegible] [illegible] node-[illegible] [illegible] [illegible] weight-[illegible] [illegible] [illegible] [illegible] bias-[illegible] [illegible] [illegible] tf.matmul(), tf.add() [illegible] [illegible]. [illegible] [illegible] [illegible] hidden [illegible] relu [illegible] activate [illegible]. [illegible] [illegible] [illegible] activate [illegible] [illegible], cost [illegible] [illegible]. [illegible] cost-[illegible] [illegible] [illegible] [illegible] [illegible] sigmoid activation [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] forward propagation [illegible] [illegible] [illegible] function-[illegible] [illegible].

[illegible] dropout optimization [illegible] [illegible] [illegible], relu activation [illegible] [illegible] [illegible] [illegible], [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible].

```
if drop_out == True:
    A = tf.nn.dropout(x = A, keep_prob = 0.8)
```

https://gist.github.com/nithyadurai87/4baef96f858284d1a4868c80df25f990

```
import numpy as np
import tensorflow as tf
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
import pandas as pd
```

```
def normalize(data):
  col_max = np.max(data, axis = 0)
  col_min = np.min(data, axis = 0)
  return np.divide(data - col_min, col_max - col_min)

(X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)
X_train, X_test, Y_train, Y_test = train_test_split(X_cancer, y_cancer,
random_state = 25)

X_train = X_train[:,:2]

import matplotlib.pyplot as plt
import matplotlib.colors
colors=['blue','green']
cmap = matplotlib.colors.ListedColormap(colors)
plt.figure()
plt.title('Non-linearly separable classes')
plt.scatter(X_train[:,0], X_train[:,1], c=Y_train, marker= 'o',
s=50,cmap=cmap,alpha = 0.5 )
plt.show()

def forwardProp(X, parameters, drop_out = False):
    A = X
    L = len(parameters)//2
    for l in range(1,L):
        A_prev = A
        A = tf.nn.relu(tf.add(tf.matmul(parameters['W' + str(l)], A_prev),
parameters['b' + str(l)]))
        if drop_out == True:
            A = tf.nn.dropout(x = A, keep_prob = 0.8)
    A = tf.add(tf.matmul(parameters['W' + str(L)], A), parameters['b' + str(L)])
    return A

def deep_net(regularization = False, lambd = 0, drop_out = False, optimizer =
False):
    tf.reset_default_graph()
    layer_dims = [2,25,25,1]
    X = tf.placeholder(dtype = tf.float64, shape = ([layer_dims[0],None]))
    Y = tf.placeholder(dtype = tf.float64, shape = ([1,None]))

    tf.set_random_seed(1)
    parameters = {}
    for i in range(1,len(layer_dims)):
        parameters['W' + str(i)] = tf.get_variable("W"+ str(i),
shape=[layer_dims[i], layer_dims[i-1]],
initializer=tf.contrib.layers.xavier_initializer(), dtype=tf.float64)
        parameters['b' + str(i)] = tf.get_variable("b"+ str(i),
initializer=tf.zeros([layer_dims[i],1],dtype=tf.float64))
```

```
    Z_final = forwardProp(X, parameters, drop_out)

    cost = tf.nn.sigmoid_cross_entropy_with_logits(logits=Z_final,labels=Y)
    if optimizer == "momentum":
        train_net =  tf.train.MomentumOptimizer(0.01,
momentum=0.9).minimize(cost)
    elif optimizer == "rmsProp":
        train_net = tf.train.RMSPropOptimizer(0.01, decay=0.999, epsilon=1e-
10).minimize(cost)
    elif optimizer == "adam":
        train_net = tf.train.AdamOptimizer(0.01, beta1 = 0.9, beta2 =
0.999).minimize(cost)
    if regularization:
        reg_term = 0
        L = len(parameters)//2
        for l in range(1,L+1):
            reg_term += tf.nn.l2_loss(parameters['W'+ str(l)])
        cost = cost + (lambd/2) * reg_term
    cost = tf.reduce_mean(cost)

    train_net = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
    init = tf.global_variables_initializer()
    costs = []
    with tf.Session() as sess:
        sess.run(init)
        for i in range(10000):
            _,c = sess.run([train_net, cost], feed_dict={X: normalize(X_train).T,
Y: Y_train.reshape(1, len(Y_train))})
            if i % 100 == 0:
                costs.append(c)
            if i % 1000 == 0:
                print(c)
        plt.ylim(min(costs)+0.1 ,max(costs), 4, 0.01)
        plt.xlabel("epoches per 100")
        plt.ylabel("cost")
        plt.plot(costs)
        plt.show()
        params = sess.run(parameters)
    return params

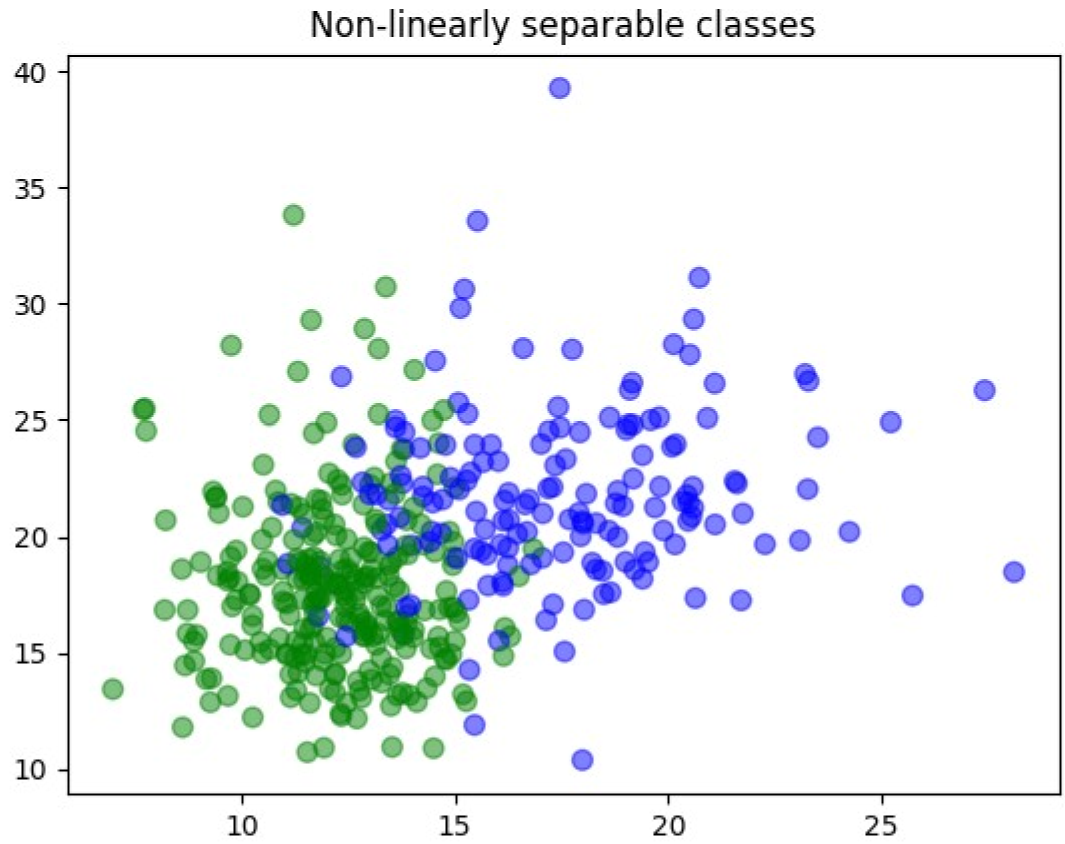
def predict(X, parameters):
    with tf.Session() as sess:
        Z = forwardProp(X, parameters, drop_out= False)
        A = sess.run(tf.round(tf.sigmoid(Z)))
    return A

def plot_decision_boundary1( X, y, model):
```

```
    plt.clf()
    x_min, x_max = X[0, :].min() - 1, X[0, :].max() + 1
    y_min, y_max = X[1, :].min() - 1, X[1, :].max() + 1
    colors=['blue','green']
    cmap = matplotlib.colors.ListedColormap(colors)
    h = 0.01
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    A = model(np.c_[xx.ravel(), yy.ravel()])
    A = A.reshape(xx.shape)
    plt.contourf(xx, yy, A, cmap="spring")
    plt.ylabel('x2')
    plt.xlabel('x1')
    plt.scatter(X[0, :], X[1, :], c=y, s=8,cmap=cmap)
    plt.title("Decision Boundary for learning rate:")
    plt.show()

p = deep_net(regularization = True, lambd = 0.02)
plot_decision_boundary1(normalize(X_train).T,Y_train,lambda x: predict(x.T,p))

p = deep_net(drop_out = True)
p = deep_net(optimizer="momentum")
p = deep_net(optimizer="rmsProp")
p = deep_net(optimizer="adam")
```



0.8640964968487806

0.6576789363554733

0.5143526817019538

0.46109938219842395

0.4391590019944309

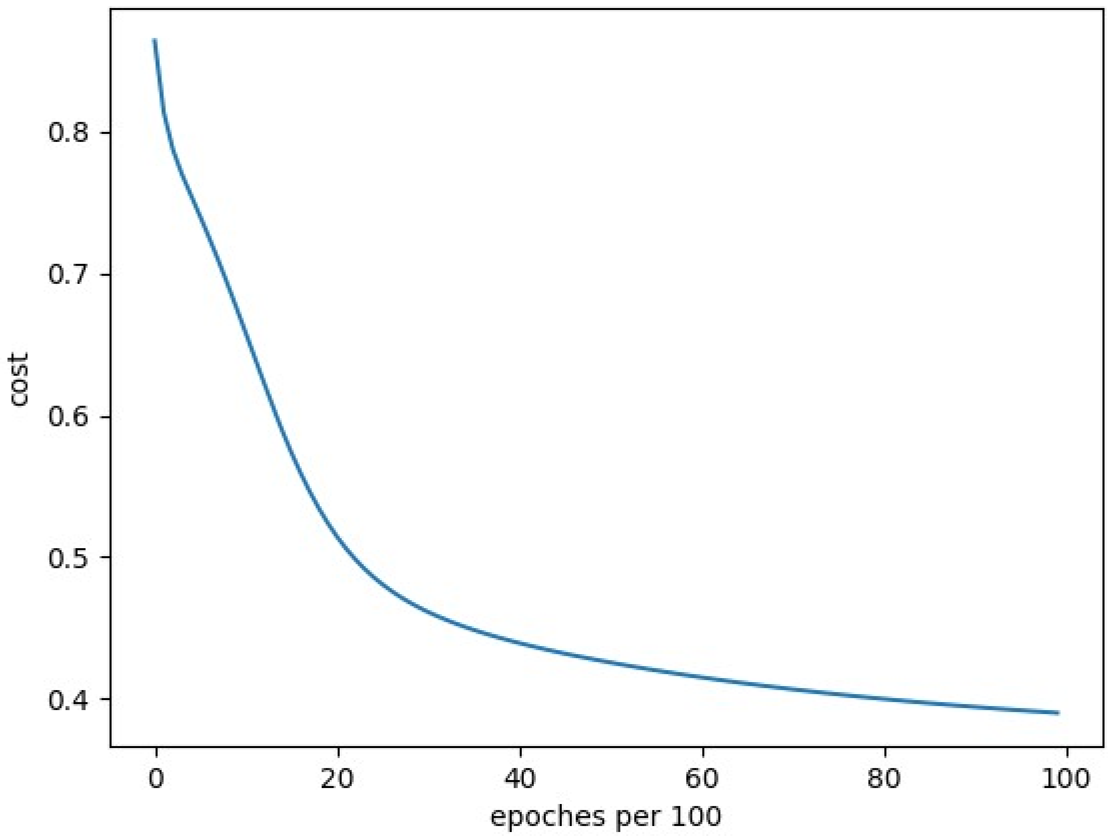
0.4253497158288082

0.41486545995180873

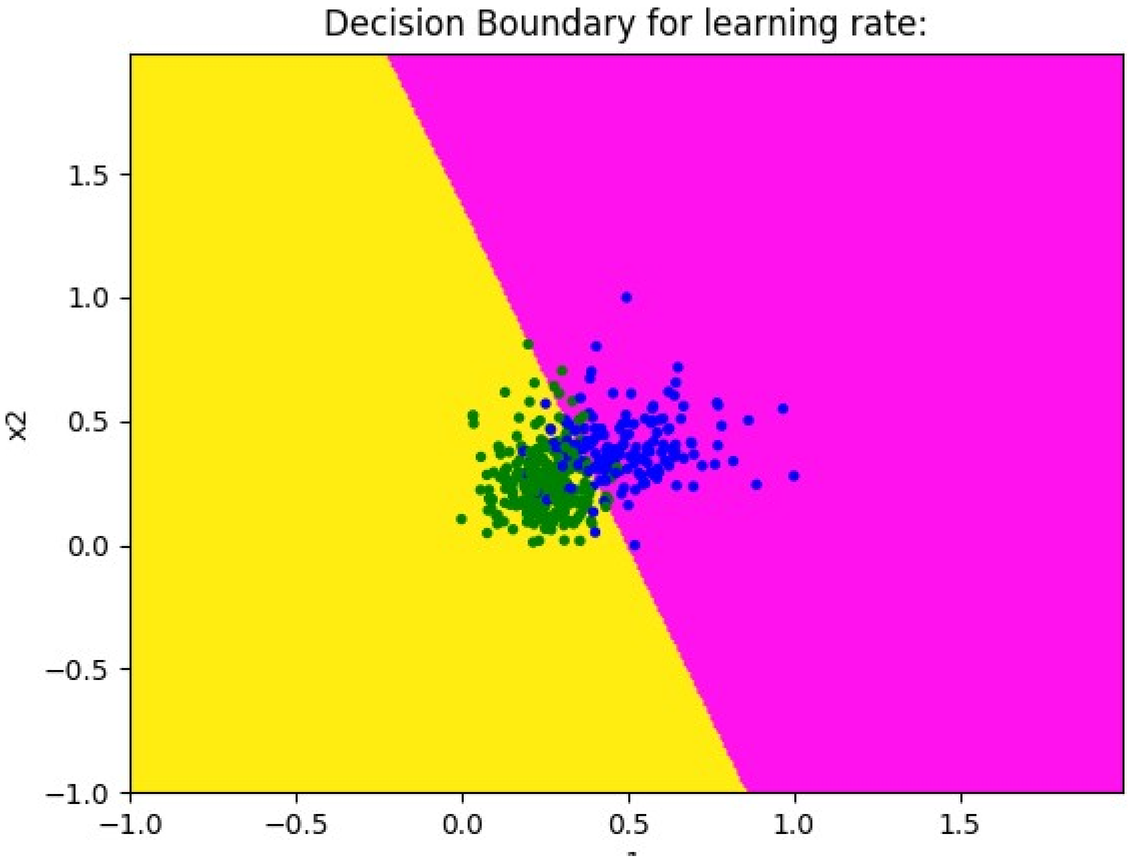
0.4064618870927915

0.39961302157076345

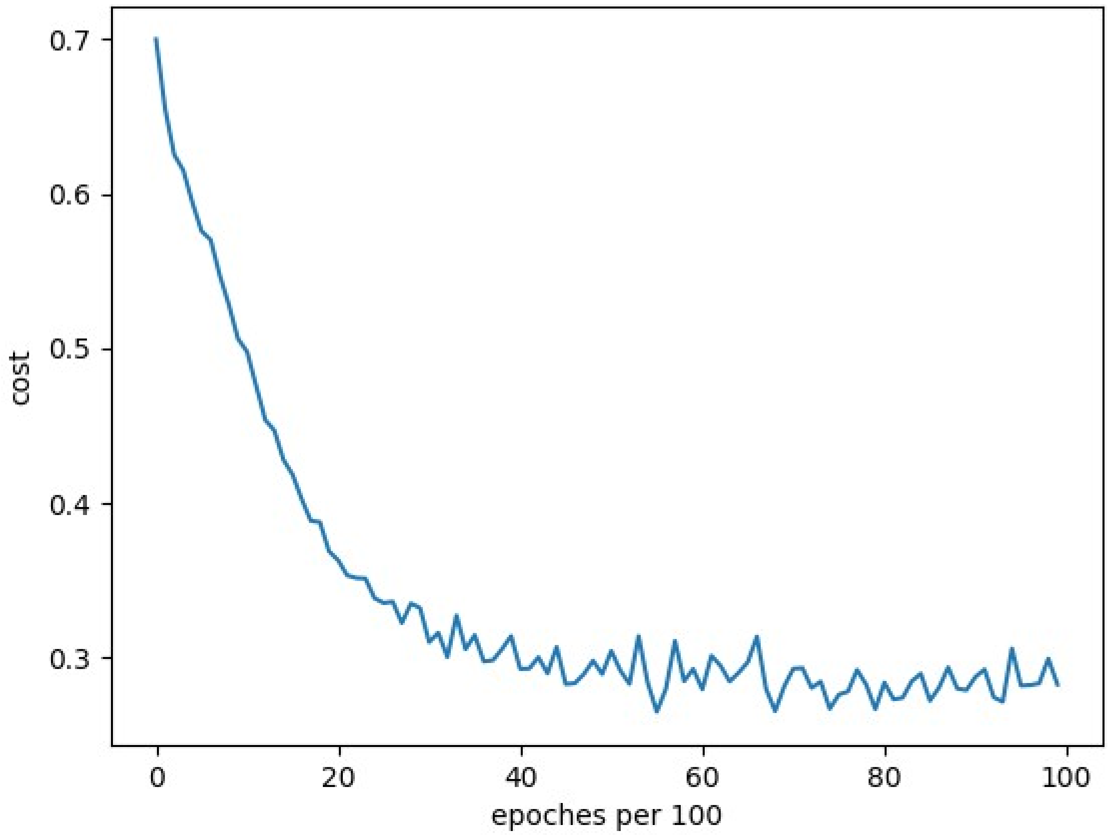
0.3940068025490444



0.8
0.7
0.6
0.5
0.4
cost
0
20
40
60
80
100
epoches per 100



0.7001094555968302
0.49777905860234667
0.36297676798298484
0.3099069894603214
0.2926005495411737
0.30422081040917726
0.279327552972575
0.2927437389848652
0.2837427041090086
0.2874399397021791



0.6979167571210456

0.46770023748893524

0.3114688059257045

0.271497538924154

0.26210240985355404

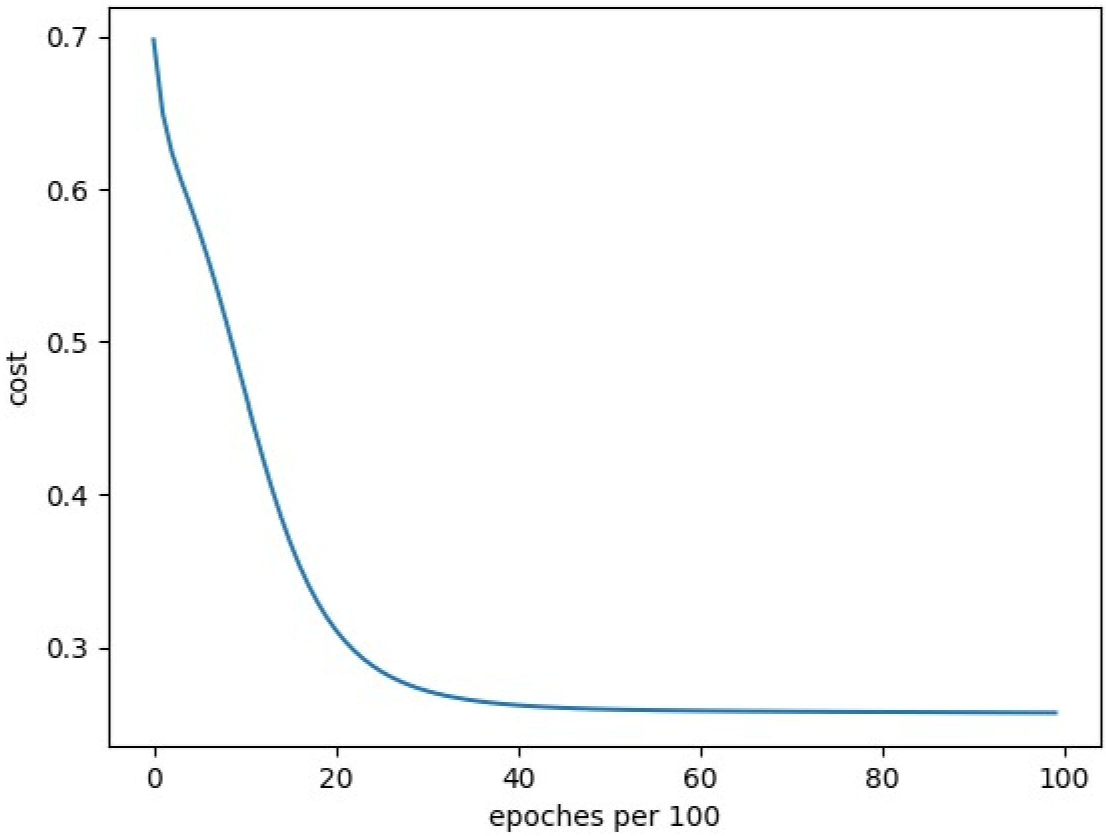
0.2593918901561954

0.25844236585729935

0.25798796421521397

0.2576690491870255

0.2573865959621496



0.6979167571210456

0.46770023748893524

0.3114688059257045

0.271497538924154

0.26210240985355404

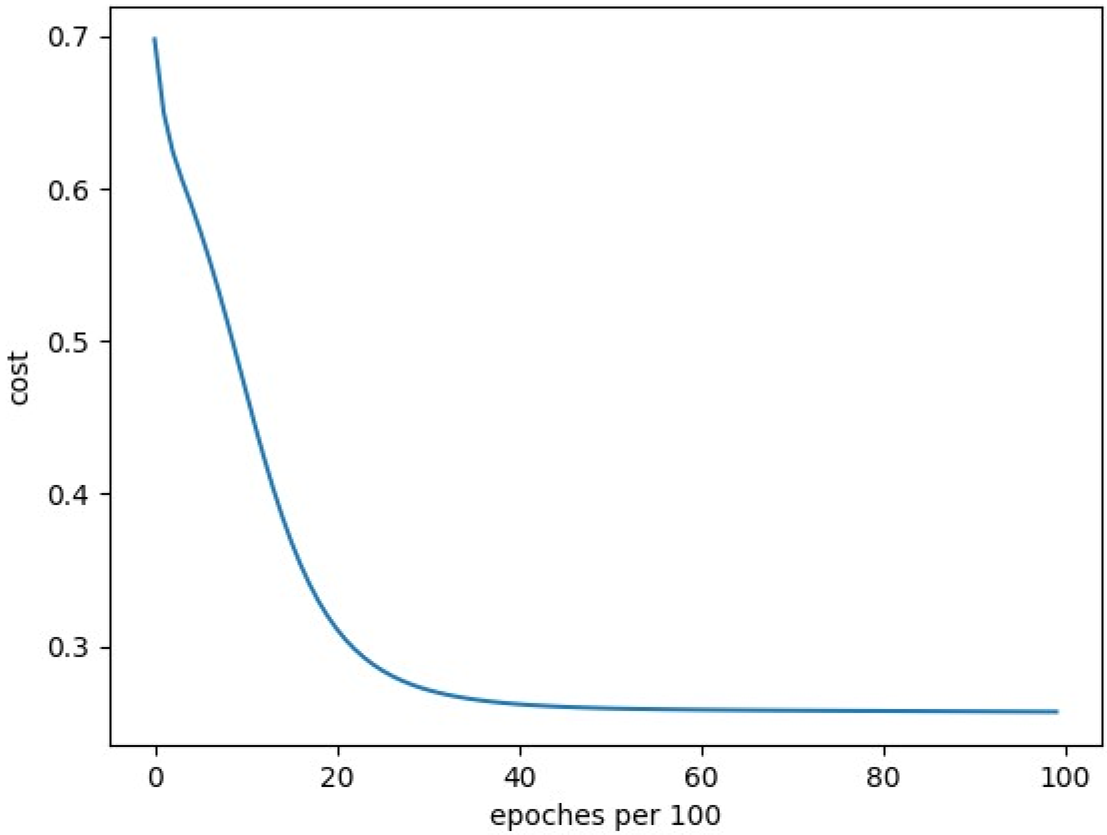
0.2593918901561954

0.25844236585729935

0.25798796421521397

0.2576690491870255

0.2573865959621496



0.6979167571210456

0.46770023748893524

0.3114688059257045

0.271497538924154

0.26210240985355404

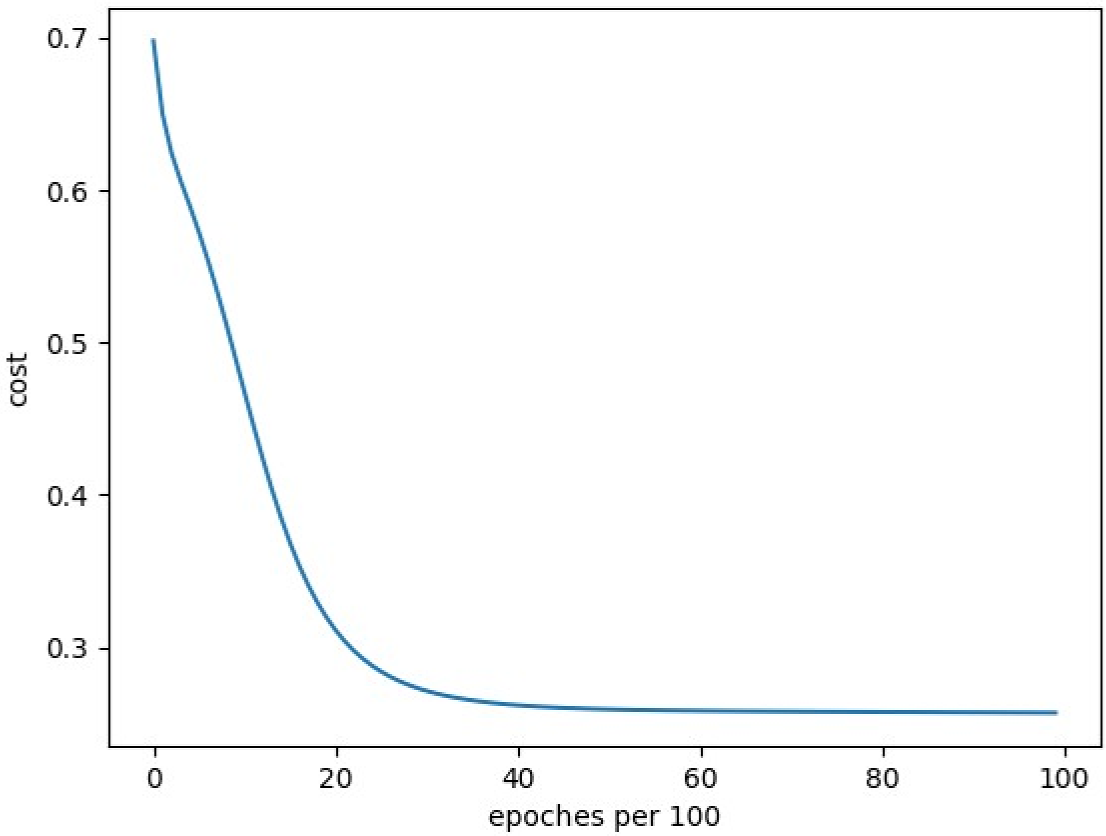
0.2593918901561954

0.25844236585729935

0.25798796421521397

0.2576690491870255

0.2573865959621496



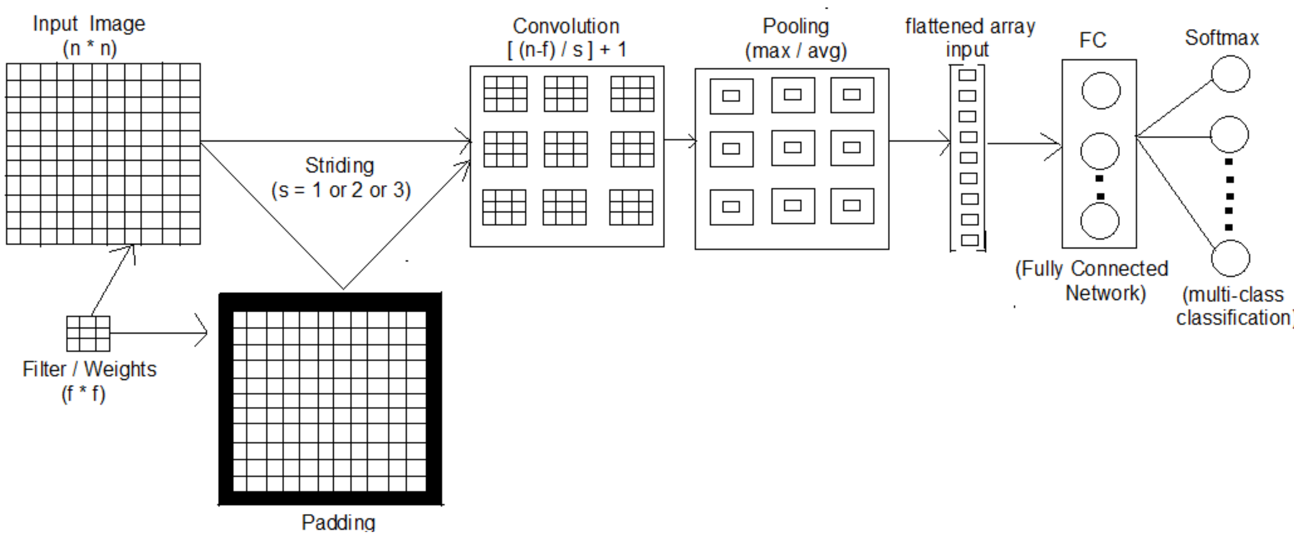
12 Convolutional Neural Networks

[illegible] database-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] 'Structured data' [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

[illegible] [illegible] [illegible] [illegible] 'structured data' [illegible], [illegible] [illegible] [illegible] [illegible], [illegible] , [illegible] [illegible], [illegible] [illegible] (images, videos, text data, voice data) [illegible] 'unstructured data’ [illegible]. CNN, RNN [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible].

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] "[illegible] [illegible]?” [illegible] [illegible] [illegible] CNN [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] (Image classification & identification) [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] CNN [illegible]. [illegible]

[illegible] Convolution, Pooling, Striding, Padding, Flattened array, FC (fully connected network) [illegible]? [illegible]? [illegible]. [illegible]. [illegible].



[illegible]1 - Input Matrix:

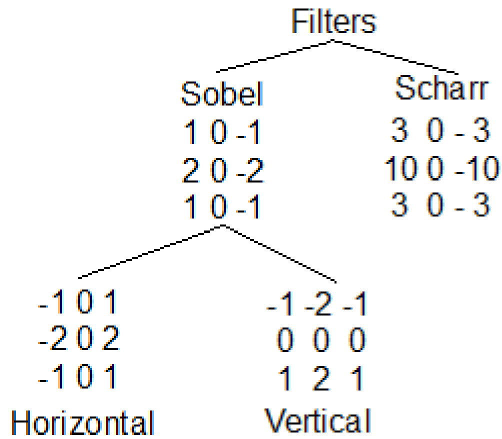
[illegible]. [illegible]. [illegible] CNN-[illegible]. RGB [illegible]. [illegible] shades-[illegible] 0-255 [illegible]. [illegible] colour codes [illegible]. [illegible]. [illegible] '[illegible]?' [illegible] features-[illegible]. [illegible] features-[illegible] CNN [illegible]. [illegible]. [illegible] Input Matrix [illegible]. [illegible] 'Convolution' [illegible] features-[illegible].

[illegible]2 - Convolution:

[illegible]. '[illegible]' [illegible]. [illegible], [illegible]. [illegible].

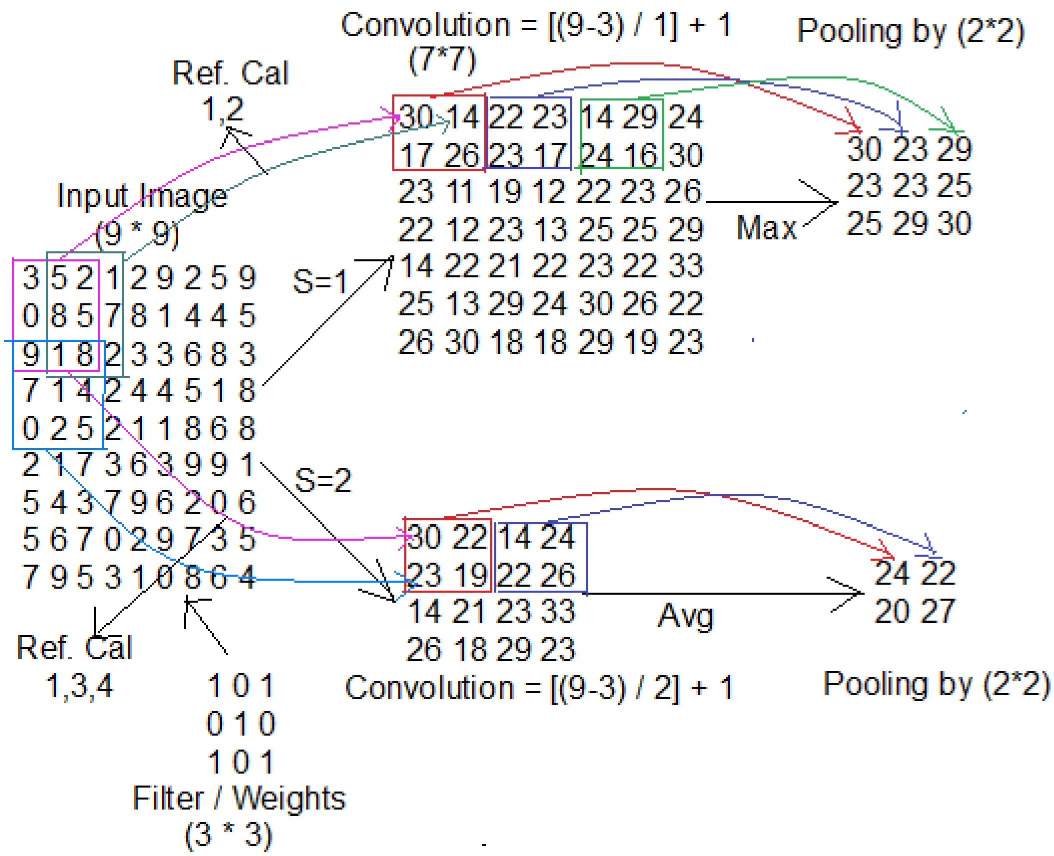
[illegible] 2*2 [illegible] 3*3 [illegible]. [illegible](filters) / [illegible](parameters) [illegible]. [illegible] weights-[illegible].

- Scharr filter, Sobel filter [illegible]. Scharr filter [illegible]. [illegible] 'Edge Detection' [illegible].
- Sobel filter [illegible]. [illegible].
- [illegible].
- [illegible]. [illegible] parameters [illegible]. [illegible] back propagation [illegible].



[illegible].

[illegible] 3,5,2,1.... [illegible], 1,0,1 [illegible] 3*3 filter [illegible]. [illegible]. [illegible] 3,5,2 [illegible] 3*3 [illegible].



[illegible]. [illegible] 30 [illegible]. [illegible] "Convolved output" [illegible].

Cal-1 30 =
3 5 2   1 0 1   3*1 5*0 2*1   3 0 2
0 8 5 * 0 1 0 = 0*0 8*1 5*0 = 0 8 0 = 3+2+8+9+8 = 30
9 1 8   1 0 1   9*1 1*0 8*1   9 0 8

Cal-2 14 =
5 2 1   1 0 1   5*1 2*0 1*1   5 0 1
8 5 7 * 0 1 0 = 8*0 5*1 7*0 = 0 5 0 = 5+1+5+1+2 = 14
1 8 2   1 0 1   1*1 8*0 2*1   1 0 2

Cal-3 22 =
2 1 2   1 0 1   2*1 1*0 2*1   2 0 2
5 7 8 * 0 1 0 = 5*0 7*1 8*0 = 0 7 0 = 2+2+7+8+3 = 22
8 2 3   1 0 1   8*1 2*0 3*1   8 0 3

Cal-4 23 =
9 1 8   1 0 1   9*1 1*0 8*1   9 0 8
7 1 4 * 0 1 0 = 7*0 1*1 4*0 = 0 1 0 = 9+8+1+0+5 = 23
0 2 5   1 0 1   0*1 2*0 5*1   0 0 5

[illegible] "Convolved output" -[illegible] stride [illegible] parameter [illegible]. [illegible] stride (S=1) [illegible] 5,2,1 [illegible]. [illegible] stride=2 [illegible] 2,1,2 [illegible]. [illegible], [illegible] "Convolved output" -[illegible]. [illegible] stride [illegible] convolution [illegible]. [illegible] n*n [illegible] f*f [illegible] [(n-f) / s] + 1 [illegible]. [illegible] 9*9 [illegible] 3*3 [illegible] convolved output-[illegible],

- stride=1 [illegible] 7*7 [illegible] i.e [(9-3) / 1] + 1,
- stride=2 [illegible] 4*4 [illegible] i.e [(9-3) / 2] + 1, [illegible].

[illegible] 'Pooling' [illegible].

3 - Pooling:

'convolution' , 'Pooling' . , . . Pooling . , , , , Max pooling, Min pooling, Avg pooling . 2*2 . 2*2 'convolved output' - .

'convolved output' - 30,14,17,26 . 'Max Pooling' 30 . 'Max pooling' .

- Padding:

. . Padding . Zero padding . , . . . Padding .. , ..

9*9 3*3 , convolved output- 7*7 . 9*9 11*11 convolved output- 9*9 i.e [(11-3) / 1] + 1 . zero padding , 9*9 convolution- 9*9 "same padding"

[illegible]. [illegible], [illegible] "valid padding" [illegible].

[illegible]:

[illegible] convolution [illegible] pooling [illegible]. [illegible] features -[illegible]. [illegible] 'Flattened array of input values' [illegible]. [illegible] FC [illegible] 'Fully connected network'-[illegible] [illegible].

[illegible]. Padding [illegible]. Convolution-[illegible]. [illegible] Pooling [illegible]. [illegible] CNN-[illegible]. [illegible]. [illegible] CNN [illegible].

https://gist.github.com/nithyadurai87/50cb753e78bba477e3ef8c01ea338c63

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

def zero_pad(X, pad):
    X_padded = np.pad(array = X, pad_width = ((0,0),(pad,pad), (pad,pad),(0,0)),
mode = 'constant', constant_values = 0)
    return X_padded

def conv_single_step(X_slice, W, b):
    conv = np.multiply(X_slice, W)
    Z = np.sum(conv)
    Z = np.add(Z, b)
    return Z

def conv_forward(X, W, b, hparams):
  stride = hparams["stride"]
  pad = hparams["pad"]
  m, h_prev, w_prev, c_prev = X.shape
```

```
  f, f, c_prev, n_c = W.shape

  n_h = int((h_prev - f + 2*pad)/stride) + 1
  n_w = int((w_prev - f + 2*pad)/stride) + 1

  Z = np.zeros((m, n_h, n_w, n_c))
  A_prev_pad = zero_pad(X, pad)
  for i in range(m):
    for h in range(n_h):
      for w in range(n_w):
        for c in range(n_c):
           w_start = w * stride
           w_end = w_start + f
           h_start = h * stride
           h_end = h_start + f
           Z[i,h,w,c] = conv_single_step(A_prev_pad[i, h_start:h_end,
w_start:w_end, :], W[:,:,:,c], b[:,:,:,c])
  return Z

def max_pool(input, hparams):
    m, h_prev, w_prev, c_prev = input.shape
    f = hparams["f"]
    stride = hparams["stride"]
    h_out = int(((h_prev - f)/stride) + 1)
    w_out = int(((w_prev -f)/stride) + 1)
    output = np.zeros((m, h_out, w_out, c_prev))
    for i in range(m):
        for c in range(c_prev):
            for h in range(h_out):
                for w in range(w_out):
                    w_start = w * stride
                    w_end = w_start + f
                    h_start = h * stride
                    h_end = h_start + f
                    output[i, h, w, c] = np.max(input[i,h_start:h_end,
w_start:w_end, c])
    assert output.shape == (m, h_out, w_out, c_prev)
    return output

img = mpimg.imread('./cake.JPG')
print (img.shape)
X = img.reshape(1,142,252,3)

fig = plt.figure(figsize=(15,10))

ax1 = fig.add_subplot(2,2,1)
print("Shape of Image: ", X.shape)
ax1.imshow(X[0,:,:,:])
```

```
ax1.title.set_text('Original Image')

ax2 = fig.add_subplot(2,2,2)
X = zero_pad(X, 10)
print("After padding: ", X.shape)
ax2.imshow(X[0,:,:,:], cmap = "gray")
ax2.title.set_text('After padding')

ax3 = fig.add_subplot(2,2,3)
W = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]]).reshape((3,3,1,1))
b = np.zeros((1,1,1,1))
hparams = {"pad" : 0, "stride": 1}
X = conv_forward(X, W, b, hparams)
print("After convolution: ", X.shape)
ax3.imshow(X[0,:,:,0], cmap='gray',vmin=0, vmax=1)
ax3.title.set_text('After convolution')

ax4 = fig.add_subplot(2,2,4)
hparams = {"stride" : 1, "f" : 2}
X = max_pool(X, hparams)
print("After pooling :", X.shape)
ax4.imshow(X[0,:,:,0], cmap = "gray")
ax4.title.set_text('After pooling')

plt.show()
```

(142, 252, 3)

Shape of Image: (1, 142, 252, 3)

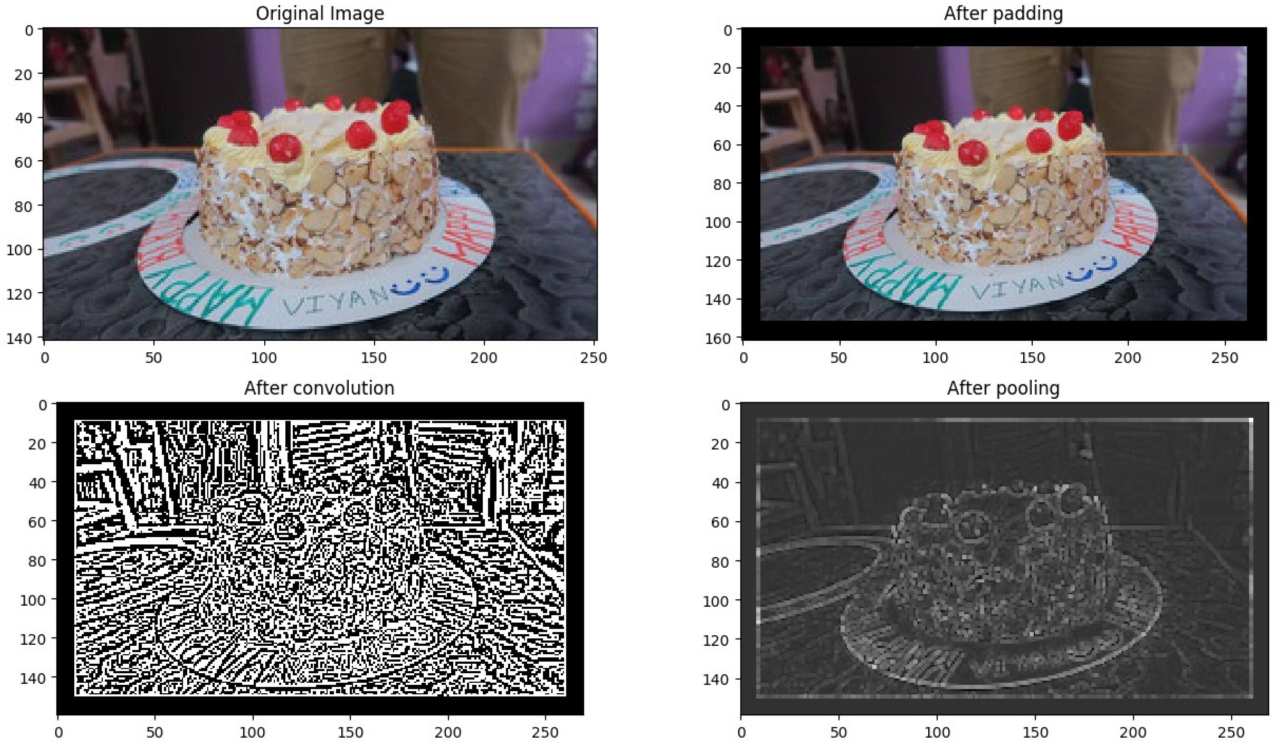
After padding: (1, 162, 272, 3)

After convolution: (1, 160, 270, 1)

After pooling : (1, 159, 269, 1)

13

14



[illegible] 'object classification' [illegible], [illegible] 'object identification' [illegible]. [illegible] AlexNet, ResNet, VGG19, InceptionResNet, GoogLeNet, DenseNet, NASNet [illegible]. [illegible].

ImageNet [illegible] database [illegible]. [illegible] ILSVRC (ImageNet Large Scale Visual Recognition Challenge) [illegible]. [illegible]. [illegible]. [illegible], [illegible].

14.1 Computer Vision

[illegible]

[illegible] [illegible] [illegible] computer vision [illegible] [illegible]. [illegible] object detection-[illegible] [illegible] [illegible] YOLO [illegible] algorithm [illegible]. You Only Look Once [illegible] YOLO [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible], [illegible] [illegible] [illegible] [illegible] grid-[illegible] [illegible]. [illegible] [illegible] grid-[illegible] [illegible] Image classification [illegible] Localization [illegible] [illegible]. [illegible] [illegible] object-[illegible] bounding boxes-[illegible] [illegible], [illegible] [illegible] class-[illegible] [illegible] [illegible] [illegible]. [illegible] object [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] bounding boxes-[illegible] [illegible] [illegible] [illegible] [illegible] Intersection Over Union [illegible] Non-max supression [illegible] [illegible] [illegible].

Face Recognition [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] app [illegible]. [illegible] [illegible] [illegible] YOLO [illegible]. [illegible] face verification [illegible] face recognition [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] face verification [illegible] (binary classification – one to one mapping).

Face recognition [illegible] [illegible] [illegible] [illegible] [illegible] db-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] face recognition [illegible]. ConvNet [illegible] [illegible] [illegible] [illegible], [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] 'One short learning’ [illegible] [illegible]. [illegible] [illegible], [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] features [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] "[illegible] [illegible] [illegible] " [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] Siamese Network [illegible].

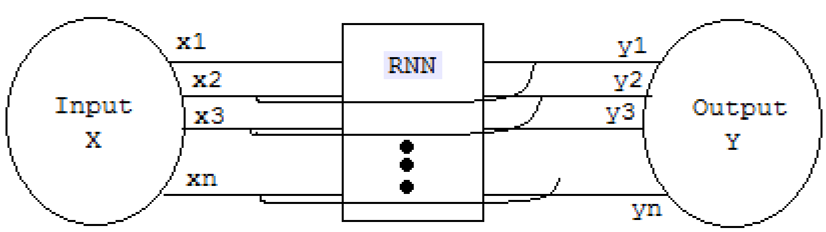
Siamese Network [illegible] similarity function-[illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] db-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] threshold-[illegible] [illegible]. [illegible] [illegible] [illegible] threshold-[illegible] [illegible] [illegible] "[illegible] [illegible]" [illegible], [illegible] [illegible] "[illegible] [illegible]" [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] triplet loss function [illegible]. Anchor img [illegible] [illegible] [illegible] [illegible]. positive ,negative images [illegible] [illegible] [illegible]

[illegible]. [illegible] anchor image, positive image, negative image [illegible] 3 [illegible]. [illegible] triplet loss [illegible].

15 Recurrent Neural Networks

[illegible] 400 [illegible], 600 [illegible]. 600 [illegible] 400 [illegible].

[illegible] 'sequential data' [illegible]. [illegible] temporal [illegible] time-component [illegible]. [illegible] RNN [illegible].



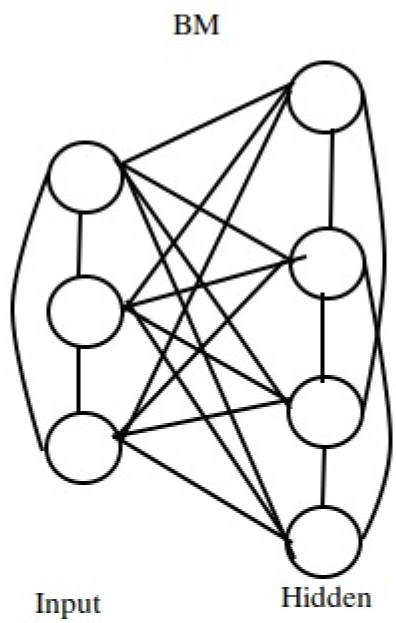
[illegible] Recurrent [illegible] memory-[illegible]. RNTN [illegible] memory-[illegible] LSTM (Long Short Term Memory) [illegible]. [illegible]-[illegible].

[illegible] 'Text Processing' [illegible]. [illegible], [illegible] RNN-[illegible], [illegible]. [illegible], [illegible], [illegible], [illegible]. [illegible], [illegible], [illegible], [illegible], [illegible] RNN-[illegible].

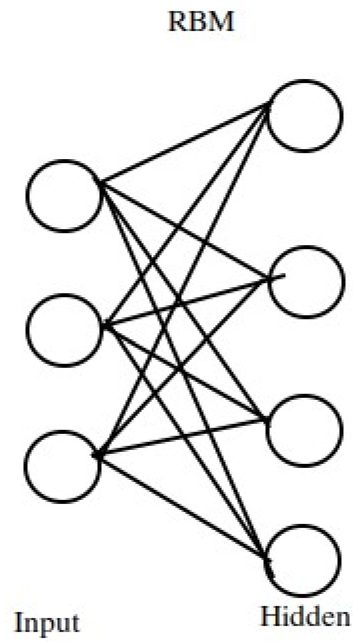
RNTN [illegible] RNN-[illegible]. [illegible], [illegible] NLP (Natural Language Processing) [illegible]. [illegible], [illegible], [illegible], [illegible] (Parts of speech tagging), [illegible] (Stemming & Lemmatization), [illegible](segmentation) [illegible] RNTN [illegible]. Multi-layer perceptron [illegible]. [illegible] RNTN [illegible].

16 BM, RBM, DBN Networks

Boltzmann Machines □□□ □□□ Restricted boltzmann machines □□□. □□□ Boltzmann Network □□□ □□□ □□□ □□□. □□□ □□□ □□□ □□□ features-□□ □□□ □□□ □□□ □□□ □□□ □□□ □□□ □□□ Boltzmann Machines □□□. □□□ □□□ input □□□ hidden □□□ □□□ □□□ □□□ □□□ □□□. □□□ deep learning □□□ □□□ □□□ output □□□ □□□ □□□ □□□ □□□. □□□ □□□ □□□ □□□ nodes □□□ undirected □□□ □□□. □□□ □□□ □□□ □□□ □□□ □□□ □□□ □□□, □□□ □□□ □□□ □□□ □□□ □□□ □□□ nodes-□ □□□. □□□ □□□ 'Generative Deep Learning Model' □□□ □□□. □□□ □□□ □□□.



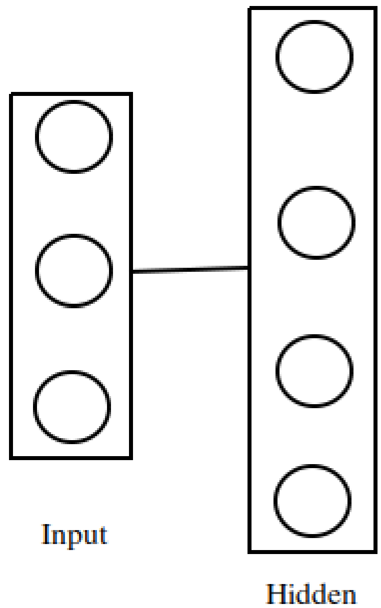
□□□ □□□ □□□ nodes-□□ □□□ □□□ □□□ □□□ □□□ feature detector-□□ □□□. □□□ □□□ □□□ □□□ □□□, □□□ □□□ □□□ nodes-□□ □□□ □□□ □□□ □□□, □□□ □□□ □□□ nodes-□□ □□□ □□□ □□□ □□□. □□□ □□□ □□□ □□□ □□□ feature detectors-□□□, □□□ □□□ □□□ □□□. □□□ □□□ □□□ Restricted Boltzmann Machines □□□. □□□ □□□ □□□.



RBM □□□□□□ □□□ □□□□□□□ □□□□ nodes □□□□□□□□□□□□ □□□□□□□ □□□□□□□□□□□□ □□□□□□□ □□□□□□□ restrict □□□□□□□□. □□□□□□□ □□□□□□□ BN-□□□ □□□□□□ □□□□□□ □□□□□□□ □□□□□ □□□□□□□ □□□□□□□□□□ □□□□□. □□□□□□□□ □□□□□ input □□□□□□, □□□□□□□□□ □□□□□ hidden □□□□□□ □□□□□□□□□□□□□. □□□ □□□ □□□□□□□ □□□□ nodes-□□□□□□ □□□□□□□□ □□□□□□□ □□□□□□□ restrict □□□□□□□□□, 'Restricted Boltzmann Network' □□□□ □□□□□□□ □□□□□□□□□□□□□□□□. □□□□□□□ undirected □□□□□□□□, □□□□□□□□□□□□ □□□□□□□ □□□□□ gradient descent □□□□□□□□ □□□□□□□□□□□ □□□ □□□□□□□ □□□□□□ □□□□□ □□□□□□□□. □□□□□□ □□□□□□ 'Contrastive divergence' □□□□□ □□□□ □□□□□□□□□□□□□□□□□□□□. □□□ □□□□□□□□ □□□□□□□□□□□ □□□□□□□□□□□□□ □□□□□□ □□□□□□□□□□ □□□ □□□□□□□□□□ □□□□□□□□□□□□ □□□□□□ □□□□□□□□□□□□□□□. □□□□ □□□ 'Energy based model' □□□□□□□ □□□□□□□□□ □□□□□□. dimensionality reduction, collaborative filtering, feature learning and topic modeling □□□□□ □□□□□□□□□ □□□ □□□□□□□□□□ □□□□□□□□. RBM □□□□□□ visible nodes □□□□□ □□□□□ □□□□□□□□□□□ □□□□□□□□□□ random -□□□□ □□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□(weights) □□□□□□□ hidden nodes-□ □□□□□□□□□□□□. □□□□□□□ □□□□□□□□□□□□□□□ hidden nodes □□□ □□□□□□□□□□□□□ □□□□□□□□□□□ □□□ □□□□□□□□□□□□□ □□□□□□ visible nodes-□ □□□ □□□□□□□□□□ □□□□□□□□□.

Deep Belief Network □□□□□□ □□□□□□□ □□□□□ □□□□□□□□□□□□□□□□ RBM □□□□□□. □□□ Stack of RBM □□□□□ □□□□□□□□□□□□□□□□□. RBM-□□ □□□□□□□ □□□□□□□□□ □□□□ □□□□□□□□ nodes-□□ □□□□□□□□□□□□ □□□□□□□□□, □□□□□ □□□□□□□□□□□□□□□□ □□□□□□□ □□□□□□□□□□□□□□□. □□□ □□□□□□□□□□□ □□□□□□□□ hidden □□□□□□□□□□ □□□□□□□□□□□□□□□, □□□□□□□ □□□□□□□ □□□□□□ □□□□□□□ □□□□□□□□□□, □□□□□□ □□□□□□ □□□□□□□□□□ □□□□□□□ □□□□□□□□□□□□□ □□□□□□

[illegible]. [illegible] [illegible] [illegible] hidden [illegible], [illegible] [illegible] [illegible] visible [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible].



[illegible] [illegible] UPN(Unsupervised Pretrained Network) [illegible] [illegible]. [illegible] [illegible] unsupervised [illegible] [illegible] [illegible] [illegible] algorithms [illegible]. RBM [illegible] features-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] features-[illegible] [illegible] [illegible]. [illegible], [illegible], [illegible] [illegible] [illegible] [illegible] cluster images, video capture sound, text [illegible] [illegible] [illegible] DBN [illegible] [illegible].

[illegible] [illegible] 943 [illegible], 1682 [illegible] [illegible] [illegible] [illegible] [illegible], [illegible] [illegible] [illegible] [illegible] [illegible]. 'u1.base' , ‘u1.test’ [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

https://www.kaggle.com/prajitdatta/movielens-100k-dataset

[illegible] [illegible] 1682 [illegible] [illegible] features-[illegible] [illegible] , [illegible] RBM [illegible] 200 features-[illegible] [illegible]. [illegible] RBM-[illegible] [illegible] [illegible] 1682 nodes-[illegible], hidden [illegible] 200 nodes-[illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] node-[illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] 943 [illegible] [illegible] [illegible] [illegible] [illegible]

□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□. □□□□□□□□□ sampling □□□□□□ 200 features
□□□□□□□□□□□□□□□□□□□□□□□□□□. □□□□□□□□□ □□□□□□ □□□□□□□□□□□□□□□.

https://gist.github.com/nithyadurai87/74b40cab9cf65929c6fe1c99c1b58a9c

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.optim as optim
import torch.utils.data
from torch.autograd import Variable

def convert(data):
    new_data = []
    for i in range(1, 944):
        id_movies = data[:,1][data[:,0] == i]
        id_ratings = data[:,2][data[:,0] == i]
        ratings = np.zeros(1682)
        ratings[id_movies - 1] = id_ratings
        new_data.append(list(ratings))
    return new_data

W = torch.randn(200, 1682)
a = torch.randn(1, 200)
b = torch.randn(1, 1682)


def hidden(x):
    wx = torch.mm(x,W.t())
    activation = wx + a.expand_as(wx)
    ph = torch.sigmoid(activation)
    return ph, torch.bernoulli(ph)

def visible(y):
    wy = torch.mm(y,W)
    activation = wy + b.expand_as(wy)
    pv = torch.sigmoid(activation)
    return pv, torch.bernoulli(pv)

def train(v0, vk, ph0, phk):
    global W,a,b
    W += (torch.mm(v0.t(), ph0) - torch.mm(vk.t(), phk)).t()
```

```
    b += torch.sum((v0 - vk), 0)
    a += torch.sum((ph0 - phk), 0)

training_set = pd.read_csv('./u1.base', delimiter = '\t')
test_set = pd.read_csv('./u1.test', delimiter = '\t')

training_set = np.array(training_set, dtype = 'int')
test_set = np.array(test_set, dtype = 'int')

print (max(max(training_set[:,0]), max(test_set[:,0])))
print (max(max(training_set[:,1]), max(test_set[:,1])))

training_set = convert(training_set)
test_set = convert(test_set)

training_set = torch.FloatTensor(training_set)
test_set = torch.FloatTensor(test_set)

training_set[training_set == 0] = -1
training_set[training_set == 1] = 0
training_set[training_set == 2] = 0
training_set[training_set >= 3] = 1
test_set[test_set == 0] = -1
test_set[test_set == 1] = 0
test_set[test_set == 2] = 0
test_set[test_set >= 3] = 1

for epoch in range(1, 11):
    train_loss = 0
    s = 0.
    for i in range(0, 943, 100):
        vk = training_set[i:i+100]
        v0 = training_set[i:i+100]
        ph0,_ = hidden(v0)
        for k in range(10):
            _,hk = hidden(vk)
            _,vk = visible(hk)
            vk[v0<0] = v0[v0<0]
        phk,_ = hidden(vk)
        train(v0, vk, ph0, phk)
        train_loss += torch.mean(torch.abs(v0[v0>=0] - vk[v0>=0]))
        s += 1.
    print('epoch: '+str(epoch)+' loss: '+str(train_loss/s))

test_loss = 0
s = 0.
for i in range(943):
    v = training_set[i:i+1]
```

```
    vt = test_set[i:i+1]
    if len(vt[vt>=0]) > 0:
        _,h = hidden(v)
        _,v = visible(h)
        test_loss += torch.mean(torch.abs(vt[vt>=0] - v[vt>=0]))
        s += 1.
print('test loss: '+str(test_loss/s))
```

[illegible]:

1. [illegible] training_set, test_set [illegible] 2 dataframes-[illegible] numpy array [illegible].

training_set = pd.read_csv('./u1.base', delimiter = '\t')

test_set = pd.read_csv('./u1.test', delimiter = '\t')

training_set = np.array(training_set, dtype = 'int')

test_set = np.array(test_set, dtype = 'int')

[illegible].

print (training_set[:3])

```
[[      1       2       3   876893171]
 [      1       3       4   878542960]
 [      1       4       3   876893119]
 [      1       5       3   889751712]]
```

- ➔ [illegible] column-[illegible] 1 [illegible] 943 [illegible].
- ➔ 2 [illegible] column-[illegible] 1 [illegible] 1682 [illegible]. [illegible], 2-[illegible] 3-[illegible] rating [illegible].
- ➔ 3 [illegible] column-[illegible]. 1 [illegible] 5 [illegible].
- ➔ 4 [illegible] column [illegible].

[illegible], [illegible]. [illegible] training, testing [illegible], [illegible], [illegible].

```
print (max(max(training_set[:,0]), max(test_set[:,0]))) - 943 Users

print (max(max(training_set[:,1]), max(test_set[:,1]))) - 1682 Movies
```

2. [illegible] RBM [illegible] 943 rows [illegible] 1682 columns-[illegible]. [illegible] convert() function [illegible].

```
def convert(data):
    new_data = []
    for i in range(1, 944):
        id_movies = data[:,1][data[:,0] == i]
        id_ratings = data[:,2][data[:,0] == i]
        ratings = np.zeros(1682)
        ratings[id_movies - 1] = id_ratings
        new_data.append(list(ratings))
    return new_data
```

[illegible] function-[illegible]. [illegible] for loop [illegible] rating [illegible], [illegible] rating [illegible] id_movies , id_ratings [illegible]. [illegible].

```
print (id_movies)

[  2   3   4   5   7   8   9  11  13  15  16  18  19  21  22  25  26  28

  29  30  32  34  35  37  38  40  41  42  43  45  46  48  50  52  55  57

  58  59  63  66  68  71  75  77  79  83  87  88  89  93  94  95  99 101

 105 106 109 110 111 115 116 119 122 123 124 126 127 131 133 135 136 137

 138 139 141 142 144 146 147 149 152 153 156 158 162 165 166 167 168 169

 172 173 176 178 179 181 182 187 191 192 194 195 197 198 199 203 204 205
```

207 211 216 217 220 223 231 234 237 238 239 240 244 245 246 247 249
251

256 257 261 263 268 269 270 271] = 134

print (id_ratings)

[3 4 3 3 4 1 5 2 5 5 5 4 5 1 4 4 3 4 1 3 5 2 1 2 3 3 2 5 4 5 4 5 5 4 5 5 4

5 2 4 4 3 4 4 4 3 5 4 5 5 2 4 3 2 2 4 5 1 5 5 3 5 3 4 5 2 5 1 4 4 3 5 1 3

3 2 4 4 3 2 5 3 4 3 4 5 5 2 5 5 5 5 5 5 3 5 4 4 5 4 4 5 5 5 4 4 5 3 5 3 5

3 3 5 1 4 2 4 4 3 2 2 5 1 4 4 4 4 1 1 5 5 5 2] = 134

□□□ □□□ 134 □□□ rating □□□. □□□ □□□ □□□ □□□ □□□ □□□ □□□. □□□ □□□ 1682 □□□ 134 □□□ rating □□□. □□□ rating □□□ □□□ 0 □□□ □□□ □□□ □□□ □□□. □□□ □□□ 0 -□ □□□ □□□ □□□ ratings □□□ □□□ □□□ 1d array □□□. □□□ □□□.

ratings = np.zeros(1682)

print (ratings)

[0. 0. 0. ... 0. 0. 0.]

□□□ 1682 columns-□□ □□□ 0’s □□□ □□□. □□□ □□□ □□□ □□□ rating □□□, □□□ □□□ □□□ 0-□ □□□ □□□ rating-□□□ □□□ □□□. □□□ ratings[id_movies - 1] = id_ratings □□□ □□□. □□□ □□□ id_movies □□□ □□□ id_movies - 1 □□ □□□ □□□. □□□ 2 □□□ □□□ 3 □□□ □□□ □□□, □□□ array-□□ □□□ □□□ 0-□ □□□ □□□, □□□ □□□ 0-□ □□□. □□□ □□□ index 1-□□□ □□□. □□□ array-□□ index 0-□□□ □□□. □□□ □□□ id_movies - 1 □□ □□□.

□□□ □□□ □□□ 943 □□□ 943 1d array □□□. □□□ new_data □□□ □□□ □□□ □□□ □□□. □□□ print(new_data[0]) □□□ □□□ □□□, □□□ □□□ rating □□□ 2,3,4,5,7 □□□ □□□ □□□ □□□ rating □□□ □□□. 1,6 □□□ □□ rating □□□ □□□ 0 □□□ □□□. □□□ □□□.

[0.0, 3.0, 4.0, 3.0, 3.0, 0.0, 4.0, 1.0, 5.0, 0.0, 2.0, 0.0, 5.0, ... 0.0, 0.0]

3. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] PyTorch [illegible] [illegible] [illegible] array [illegible] [illegible] [illegible] [illegible].

```
training_set = convert(training_set)
test_set = convert(test_set)
training_set = torch.FloatTensor(training_set)
test_set = torch.FloatTensor(test_set)
```

4. [illegible] binary classification-[illegible] problem [illegible] [illegible] [illegible] 0's & 1's [illegible] [illegible]. 1,2 [illegible] [illegible] [illegible] 0-[illegible], 3,4,5 [illegible] [illegible] [illegible] [illegible] [illegible] 1- [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] 0 [illegible] -1 [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible].

```
training_set[training_set == 0] = -1
training_set[training_set == 1] = 0
training_set[training_set == 2] = 0
training_set[training_set >= 3] = 1
test_set[test_set == 0] = -1
test_set[test_set == 1] = 0
test_set[test_set == 2] = 0
test_set[test_set >= 3] = 1
```

5. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] 943 rows [illegible] 1682 columns [illegible].

```
print (training_set)
print (training_set.size())
tensor([[-1.,  1.,  1.,  ..., -1., -1., -1.],
        [ 1., -1., -1.,  ..., -1., -1., -1.],
        [-1., -1., -1.,  ..., -1., -1., -1.],
        ...,
        [ 1., -1., -1.,  ..., -1., -1., -1.],
```

```
        [-1., -1., -1.,  ..., -1., -1., -1.],
        [-1.,  1., -1.,  ..., -1., -1., -1.]])
torch.Size([943, 1682])
```

[illegible] for loop [illegible] 10 [illegible] [illegible], [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] for loop [illegible] [illegible] [illegible] [illegible] 10 [illegible] [illegible] [illegible] [illegible] [illegible]. weights, bias [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] random-[illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] train() [illegible] function-[illegible] [illegible]. Input [illegible] hidden [illegible] [illegible] [illegible] weights [illegible] [illegible]. [illegible] [illegible] bias [illegible] [illegible] [illegible]. a [illegible] hidden [illegible] . b [illegible] input [illegible]. weights [illegible] random-[illegible] [illegible] (200,1682) [illegible] [illegible] [illegible] [illegible] [illegible].

hidden() [illegible] visible() functions [illegible] [illegible] [illegible] hidden node-[illegible] [illegible], visible node-[illegible] [illegible] [illegible]. [illegible] [illegible] 2 [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] sample-[illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] Bernoulli [illegible] [illegible] sample-[illegible] [illegible]. [illegible] hidden () [illegible] [illegible] [illegible] torch.Size([100, 200]) [illegible] [illegible], visible () [illegible] [illegible] [illegible] torch.Size([100, 1682]) [illegible] [illegible] [illegible].

```
print (ph0)
tensor([[9.9967e-01, 2.0279e-21, 1.0000e+00,  ..., 3.9762e-26, 1.0000e+00,
1.0000e+00],
        [1.1460e-09, 2.4450e-22, 1.0000e+00,  ..., 3.7041e-22, 1.0000e+00,
2.0576e-08],
        [1.9920e-03, 1.8334e-11, 1.0000e+00,  ..., 3.2599e-21, 1.0000e+00,
1.1196e-17],
        ...,
        [3.3063e-03, 4.9096e-21, 1.0000e+00,  ..., 1.1215e-25, 1.0000e+00,
2.0377e-05],
        [9.8176e-01, 5.9265e-14, 1.0000e+00,  ..., 6.0945e-25, 1.0000e+00,
2.3354e-03],
```

[2.9210e-09, 1.6359e-11, 1.0000e+00, ..., 2.2869e-15, 1.0000e+00, 2.2399e-13]])

[illegible] RBM, features-[illegible]. [illegible], [illegible] hidden [illegible].

17 Autoencoders

Autoencoder [illegible]. [illegible] dimensionality reduction, feature representation [illegible]. Machine learning-[illegible] PCA-[illegible], [illegible] autoencoder [illegible]. PCA [illegible] dimension-[illegible] , Autoencoder [illegible] [illegible](non-linear) [illegible]. [illegible] unsupervised learning [illegible]. [illegible], [illegible].

[illegible]. [illegible] encoding [illegible] decoding [illegible]. [illegible]. [illegible]. [illegible], [illegible]. [illegible] encoding [illegible] decoding [illegible] 'bottleneck' [illegible]. [illegible] (denoising image) , [illegible] (dimensionality reduction), [illegible] (Image colouring) , [illegible] (feature variation) , [illegible], [illegible] (watermark removal)[illegible].

- Denoising Autoencoder [illegible]. [illegible] / [illegible] (robust features) [illegible] , [illegible].
- Sparse Autoencoder [illegible] nodes-[illegible], [illegible] nodes-[illegible]. [illegible] encode [illegible] node-[illegible]. [illegible] nodes-[illegible] Information bottleneck-[illegible].
- Deep Autoencoder [illegible], [illegible], [illegible] features-[illegible]. [illegible], [illegible]. Image Search, Data compression, Topic Modeling, Information retrieval [illegible].
- Contractive autoencoder [illegible] (unlabeled data) [illegible] label-[illegible]. [illegible] latent variables [illegible]. [illegible] Regularized autoencoder-[illegible].
- Variational autoencoder [illegible]. [illegible]! [illegible], [illegible]! [illegible] Variational autoencoder-[illegible]. Graphical models [illegible] image generation [illegible].

18 Reinforcement Learning

[illegible] 'Reinforcement Learning' [illegible]. [illegible] (self-driving cars), [illegible] gaming industry [illegible]. [illegible] 'Deepmind' [illegible]. [illegible] Reinforcement Learning [illegible].

[illegible]. [illegible] learning [illegible]. [illegible], [illegible] 'Reinforcement Learning' [illegible]. [illegible] reinforcement learning-[illegible]. [illegible] Agent, Environment, State, Action, Reward, Penalty, Policy [illegible].

1. [illegible] 'Agent' [illegible]. [illegible] 'Agent' [illegible].

2. [illegible] 'Environment' [illegible]. [illegible], [illegible] 60 [illegible], [illegible], [illegible] environment [illegible].

3. Environment-[illegible] 'state' [illegible]. [illegible] 'state' [illegible]. [illegible] 'action' [illegible]. [illegible] environment-[illegible] state-[illegible], agent [illegible] action-[illegible].

4. Agent [illegible] action [illegible] reward-[illegible], [illegible] penalty-[illegible]. [illegible] '[illegible]' [illegible] '[illegible]' [illegible]. [illegible], [illegible] 'reward' [illegible] 'penalty' [illegible]. [illegible], [illegible]. [illegible], [illegible].

5. [illegible]. [illegible]. 5-[illegible]. [illegible] 'strategies' [illegible]. [illegible] optimized strategy-[illegible] Agent [illegible]. [illegible] strategies-[illegible] 'Policy' [illegible].

[illegible] 'Reinforcement Learning' [illegible] supervised [illegible] Unsupervised [illegible]. [illegible] unsupervised [illegible], [illegible] agent-[illegible] supervised [illegible]. [illegible],, Chess [illegible], [illegible] reinforcement [illegible]. [illegible], [illegible], [illegible]. [illegible].

- ➔ [illegible] (Observation of Environment)
- ➔ [illegible] Agent [illegible] (Act according to the current state)

- [illegible] (Getting reward or penalty)
- Agent [illegible] (Finding a strategy from experiences)
- Agent [illegible] action [illegible] agent [illegible] reward [illegible] penalty [illegible]. (Iterate until finding a best strategy)

[illegible] Agent-[illegible]. (Don’t be greedy). [illegible]. (sequence of states according to time)

[illegible] reinforcement learning [illegible] Gym [illegible] module -[illegible].

https://gist.github.com/nithyadurai87/d25a4bdf226a7b5df92267e23ce8d28e

```
import gym
import numpy as np
import random
```

```
e = gym.make("Taxi-v3").env
e.render()

e.reset()
e.render()

print (e.action_space)
print (e.observation_space)

e.s = e.encode(3, 1, 2, 0) # (taxi row, taxi column, passenger index, destination
index)
e.render()

print (e.s)
print (e.P[328])

epochs, penalties, reward = 0, 0, 0
while not reward == 20:
    action = e.action_space.sample()
    state, reward, done, info = e.step(action) # (284, -1, False, {'prob': 1.0})
    if reward == -10:
        penalties += 1
    epochs += 1
print("Timesteps taken:",epochs)
print("Penalties incurred:",penalties)

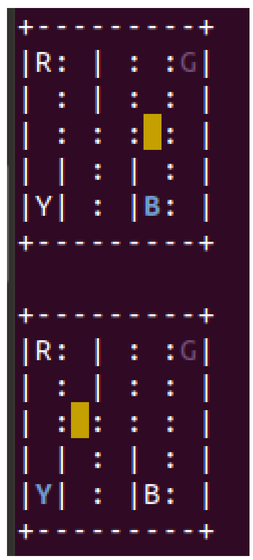
q_table = np.zeros([e.observation_space.n, e.action_space.n])
print (q_table[328])
total_epochs, total_penalties = 0, 0
for i in range(1, 100):
    state = e.reset()
    epochs, penalties, reward, = 0, 0, 0
    done = False
    while not done:
        if random.uniform(0, 1) < 0.1:
            action = e.action_space.sample() # Explore action space
        else:
            action = np.argmax(q_table[state]) # Exploit learned values
        next_state, reward, done, info = e.step(action)
        old_value = q_table[state, action]
        next_max = np.max(q_table[next_state])
        new_value = (1 - 0.1) * old_value + 0.1 * (reward + 0.6 * next_max)
        q_table[state, action] = new_value
        if reward == -10:
            penalties += 1
        state = next_state
        epochs += 1
```

```
    total_penalties += penalties
    total_epochs += epochs

print (q_table[328])
print("Timesteps taken:",epochs)
print("Penalties incurred:",penalties)
print("Average timesteps per episode:",(total_epochs / 100))
print("Average penalties per episode",(total_penalties / 100))
```

[illegible]:

1. gym.make("Taxi-v3").env [illegible]. [illegible] render() [illegible] reset() [illegible]. [illegible].



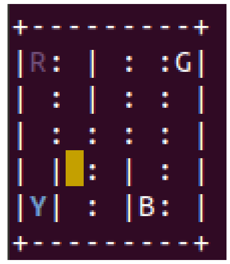
- Agent : [illegible] Agent. [illegible].
- Environment : [illegible] 5*5 [illegible]. [illegible] R, G, Y, B [illegible]. [illegible] (0,0) , (0,4) , (4,0) , (4,3) [illegible].

- Episode : [illegible] [illegible] [illegible] Agent [illegible] 'Action' [illegible] [illegible] action-[illegible] [illegible] [illegible]. Agent-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] Agent [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] 'episode' [illegible] [illegible].

- Action Space : [illegible] [illegible] [illegible] 6 [illegible] [illegible] [illegible]. [illegible] [illegible], [illegible], [illegible], [illegible] [illegible] [illegible] [illegible] [illegible] 0,1,2,3 [illegible], [illegible] [illegible] [illegible], [illegible] [illegible] [illegible] 4, 5 [illegible] [illegible]. [illegible] 'Action Space’ [illegible] [illegible]. print (e.action_space) [illegible] 6 [illegible] [illegible] [illegible] [illegible].

- State Space : [illegible] [illegible] [illegible] [illegible] [illegible] Agent [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] ' State Space’ [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] 5 [illegible] 5 [illegible] (R, G, Y, B [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] 5 [illegible] [illegible]). [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. 4 [illegible] [illegible] 4 [illegible] [illegible]. [illegible] 5*5 [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] 25 [illegible] [illegible] [illegible] [illegible] [illegible], 25 [illegible] [illegible]. [illegible] 5 [illegible], 4 [illegible], 25 [illegible] [illegible] [illegible] [illegible] state space 500 (5 * 4 * 25) [illegible] [illegible]. print (e.observation_space) [illegible] 500 [illegible] [illegible] [illegible] [illegible].

2. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible], [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. e.s = e.encode(3, 1, 2, 0) [illegible] [illegible] [illegible] [illegible] (3,1) [illegible] [illegible]. [illegible] [illegible] 2, 0 [illegible] [illegible] passenger index [illegible] destination index-[illegible] [illegible]. e.render() [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].



3. print (e.s) [illegible] 328 [illegible] [illegible] [illegible]. [illegible] [illegible] 500 [illegible] [illegible] 328-[illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] print (e.P[328]).

{0: [(1.0, 428, -1, False)],

1: [(1.0, 228, -1, False)],

2: [(1.0, 348, -1, False)],

3: [(1.0, 328, -1, False)],

4: [(1.0, 328, -10, False)],

5: [(1.0, 328, -10, False)]}

[illegible] [illegible] {action1: [(probability, nextstate, reward, done)]} [illegible] [illegible] [illegible].

[illegible] [illegible] 0,1,2,3,4,5 [illegible] Agent [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] 1.0 [illegible] Agent [illegible] [illegible] [illegible] probability-[illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] 1 [illegible] [illegible]. [illegible] [illegible] 428, 228, 348...[illegible] [illegible] Agent [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] 3 [illegible] [illegible] [illegible] [illegible], 4, 5 [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] 328 [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] -1, -10 [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible], [illegible] [illegible] [illegible] True [illegible] [illegible] 20 [illegible] [illegible]. [illegible] [illegible] [illegible] False [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible], [illegible] True [illegible], [illegible] [illegible] [illegible] [illegible] 20 [illegible] [illegible]. [illegible] [illegible] 'Episode' [illegible] [illegible].

4. [illegible] RL-[illegible] [illegible] [illegible] algorithm-[illegible] [illegible], random-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] loop [illegible] [illegible]. [illegible] [illegible] [illegible] drop-off [illegible] [illegible] [illegible] [illegible]

20 □□ □□□□□□ □□□ □□□□□□□ □□□□□□□□□ destination-□ □□□□□□□□ □□□□□□□ □□□□□□□□□ while not loop □□□□□. e.action_space.sample() □□□□□□ □□□□□ □□□□□□□□□□ □□□□□□□ random-□□ □□□ □□□□□ □□□□□□□ □□□□□□□□□□□□□.
e.step(action) □□□□□□ □□□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□□ □□□□□□ □□□□□□□□□ □□□□□□□, □□□□□□□□□□□□□□□ □□□□□□□□□ □□□□□□□ □□□□□□□, □□□□□□□□□ □□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□ □□□□□□ (284, -1, False, {'prob': 1.0}) □□□□ □□□□□□□ □□□□□□□□□□□□□□□□□. □□□□□ □□□□□ □□□□□□ □□□□□□□□□ state, reward, done □□□□□ variables-□□ □□□□□□□□□□□□□□□□□□□. □□□□□□□ □□□□ □□□□□□□ □□□□□ □□□□□□□□□□□□□□□□□□, penalty □□□□□□□□□□ 1 □□□□□□□□□□□□□□□. □□□□□□□□□□ □□□□□ □□□□□□□□□□□ □□□□□□ □□□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□ □□□□□□□ 20 □□ □□□□ loop-□ □□□□□□ □□□□□□□□□□□□□□. □□□□□□□ □□□ episode □ □□□□□□□□□□□□ □□□□□□□ □□□□□□ □□□□ □□□□□□□□□□□□ □□□□□□□□□□, □□□□□□ □□□□ □□□□□ □□□□□ □□□□□□□□ □□□□□□ penalties □□□□□□□□□□□□ □□□□□□□□□□ □□□□□□□□□□□□□□□□□□.

```
epochs, penalties, reward = 0, 0, 0
while not reward == 20:
    action = e.action_space.sample()
    state, reward, done, info = e.step(action)
    if reward == -10:
        penalties += 1
    epochs += 1
```

Timesteps taken: 1485

Penalties incurred: 473

5. Q-Learning Algorithm: □□□□□□□□ □□□□□□□ □□□□□□□ 1485 □□□□ □□□□□□□□□□□□□□□, 473 □□□□ □□□□□ □□□□□□ penalties □□□□□□□□□□□□□□□ □□□□□□□. □□□□□□□ Q-Learning □□□□□ RL-□□□□□ algorithm-□□□ □□□□□□□□□□□□□□□, □□□□□□□□□□□□ □□□□□□□□□ □□□□□□□. □□□ Agent □□□□□□□□□□□ □□□□□□□□ □□□□□□□□□ □□□□□□□ □□□□□□□□□□□ □□□□□□□ □□□□□□□□□ □□□□□□□ □□□□□□□□ □□□□□□ □□□□□□□ □□□□□□□□□□□□ □□□□□□□□□□□ □□□□□ □□□□□□□□ □□□□□□□ □□□□□□□□. □□□□□□□□ memory-□ Agent-□□□□ □□□□□ □□□□ algorithm □□□□□□□□□□□□□ Q-table □□□□□. □□□□□ □□□□□□ □□□□□□□□□□□ □□□□□□□ □□□□□□□□□□ □□□□□□□□□□ □□□□ □□□□□□□□ □□□□□□□□□ □□□□□ □□□□□□□ □□□□□□□□□□□□□□□□□. □□□□□ Q □□□□□□□ □□□□□. □□□

□□□□□□□ □□□□□□□ □□□□□□□□□□ □□□□□□□□□ □□□□□□□.

Q-table □□□□□□ State * Action □□□□□ □□□ □□□□ □□□□□□□□□□ □□□□□□□. □□□□□□ □□□□□□□ □□□□ 500 □□□□□□□□□ states □□□□□□, □□□□□□□ □□□□□□□□□□□ 6 □□□□□□□□□□ □□□□□□□□□□□□ columns-□□□□□□ □□□□□□□. □□□□□□□□□□□□□□□□□□□ Q □□□□□□□□□□ □□□□□□□□□□□□□□□□□□□. □□□□□□□ □□□□□□□□□□ □□□□ □□□□□□□□□□□ Q □□□□□□□ □□□□□□□ □□□□□□ □□□□□ □□□□□□□□□□ □□□□□□□□ □□□□□□□□ □□□□□□□□□□□ .

6. □□□□□□□□ □□□□□□□ □□□□□□□ 0 □□□□□□□□□□□ □□□□□□ □□□□□□□□□ Q-table □□□□□□□□□□□□□□□□□□. □□□□ dimension 500 * 6 □□ □□□□□□□□□□□□□ □□□□□□□□□□□ □□□□□□□□□□□□□□□□. □□□□□□□ 328-□□□ □□□□□□□□□ □□□□□□□ □□□□□□□□□□ 0 □□□□□ □□□□□□ □□□□□□□□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□□.

```
q_table = np.zeros([e.observation_space.n, e.action_space.n])

print (q_table[328])

[0. 0. 0. 0. 0. 0.]
```

□□□□□□□□□□□□ update □□□□□□□□ □□□□□□□□□□ 100 □□□□ for loop □□□□□ □□□□□□□□□□□□□□□□□. □□□□□□ 100 □□□□ □□□□□□□□□□ □□□□□□□□ □□□□□□ □□□□□□□□□ □□□□□□□□□□□□ Q □□□□□□□□□□ □□□□□□□ □□□□□□□□ □□□□□ update □□□□□□□□□□□□. □□□□□□□ □□□□□□□ □□□□□□ □□□□□□□ □□□□ Q □□□□□□□□□ □□□□□□ □□□□□□ □□□□□□□□□ □□□□□□□□□□□□□□□. □□□□□□□□ 100 episodes □□□□□□□□□□ □□□□□□□ □□□ □□□□□□□□□ Q □□□□□□□ □□□□□□□□□□□□ □□□□□□□□ □□□□□□□.

[-1.12496344 -1.12745359 -1.09896023 -1.0872184 -2.80419013 -2.78076376]

□□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□□□□□ □□□□ □□□□□□□□□ (-1.0872184)□□□□□□ □□□□□□□□□ 3-□□□ □□□□□ □□□□□□□□□□□□□ □□□□□□□□□□ □□□□□□ □□□□□□□□□ □□□□□□□□□. □□□□□ □□□□□□□□ □□□□□□□ episodes-□ □□□□□□□□□ □□□ □□□□□□□□□□ □□□□□□□□□□. □□□□□ learning by greedy □□□□□□□. □□□□□□□ □□□□□□ □□ □□□□□□ □□□□□□□□□□□□□□□ □□□□□□□□□□ □□□□□□□ □□□□□□□□ □□□□□□□□□□□□ □□□□□□□ □□□□ □□□□□□□□□□ □□□□□□□ □□ □□□□□□□□. □□□□ □□□□□□□ □□□□□ 100-□□□ episode-□□ □□□□□□□□□ □□□□□□ □□□□□□□□□□□□□□, □□□□□□□ penalty □□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□. □□□□□□ □□□□□□□□□ □□□ episode-□□□□ □□□□□□□ □□□□□□□□□□ □□□□□□□□ □□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□. □□□□□□□□□□□□ □□□□□□ □□□□□□□□ □□□□□□□□□□□□□ □□□□□□□□□□ □□□□□ □□□□□□□ □□□□□□□□ □□□□□□□. □□□□□ □□□□□ Q-algorithm □□□□□ □□□□□□□□ □□□□□□□□□ □□□□□□□ □□□□□□ □□□□□□□ □□□□□□□□□□□□□□□.

Timesteps taken: 331

Penalties incurred: 16

Average timesteps per episode: 452.2

Average penalties per episode 24.67

7. [illegible] [illegible] Q-Table [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

```
new_value = (1 - alpha) * old_value + alpha * (reward + gamma * next_max)
```

[illegible] alpha, gamma [illegible] hyperparameters . Alpha [illegible] [illegible] [illegible] [illegible] learning rate-[illegible] [illegible]. [illegible] 0.1 [illegible] [illegible] .

gamma [illegible] 0 [illegible] 1 [illegible] [illegible] [illegible] [illegible]. [illegible] 0.6 [illegible] [illegible] . [illegible] [illegible] 0 -[illegible] [illegible] [illegible], quick rewards-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. 1-[illegible] [illegible] [illegible], long-term rewards-[illegible] [illegible] [illegible].

[illegible] epsilon [illegible] [illegible] parameter [illegible] overfitting-[illegible] [illegible] [illegible]. for loop-[illegible] [illegible], [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] random-[illegible] [illegible] [illegible] [illegible] [illegible] Q [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] Q [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] over fitting [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] epsilon-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] random-[illegible], [illegible] [illegible] [illegible] Q [illegible] [illegible] [illegible] [illegible].

19 [illegible]

[illegible] Deep Learning [illegible] [illegible].[illegible]. AI, Neural Networks [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible], [illegible], StackOverFlow [illegible] [illegible], [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

[illegible]

20 [illegible]

http://freetamilebooks.com/authors/nithyaduraisamy/



21 [illegible]

[illegible]

[illegible]

- [illegible]
- [illegible]
- [illegible]
- [illegible]

[illegible]

- [illegible]
- [illegible]
- [illegible]
- **editor@kaniyam.com** [illegible]

  - [illegible]**:** [illegible]
  - [illegible]

    - [illegible]
    - [illegible]

- [illegible], [illegible].

- [illegible].
- [illegible], [illegible].
- [illegible].
- [illegible], [illegible], [illegible], [illegible], [illegible], [illegible].
- [illegible].
- [illegible] editor@kaniyam.com [illegible].
- [illegible], [illegible].
- [illegible] editor@kaniyam.com [illegible].

[illegible]

- [illegible].
- [illegible].
- [illegible].
- [illegible].
- [illegible].

[illegible]

[illegible] [illegible] [illegible] [illegible].

[illegible],
[illegible] [illegible] [illegible] [illegible] [illegible]
[illegible] [illegible] [illegible], [illegible], [illegible], [illegible],
[illegible] [illegible] [illegible], [illegible] [illegible] [illegible] [illegible]
[illegible].

[illegible]: [illegible]. [illegible] – editor@kaniyam.com +91 98417 95468

[illegible] [illegible] [illegible] [illegible]
[illegible]

22 [illegible]

22.1 [illegible] – Vision

[illegible]

22.2 [illegible] – Mission

[illegible]

22.3 [illegible]

- [illegible] – kaniyam.com
- [illegible] – FreeTamilEbooks.com

22.4 [illegible]

- [illegible] – Text to Speech
- [illegible] – Optical Character Recognition
- [illegible]
- [illegible] – Send2Kindle
- [illegible]
- [illegible]
- [illegible] – [illegible]
- [illegible] – [illegible]
- FreeTamilEbooks – [illegible]
- FreeTamilEbooks – [illegible]
- WikisourceEbooksReport [illegible]
- FreeTamilEbooks.com – Download counter [illegible]

22.5 [illegible]/[illegible]

- [illegible]/[illegible]
- [illegible]
- [illegible] NLP [illegible]
- [illegible]
- [illegible], [illegible]
- [illegible], [illegible]
- [illegible]
- [illegible]
- [illegible]
- OpenStreetMap.org [illegible], [illegible], [illegible]
- [illegible] OpenStreetMap.org [illegible]
- [illegible]
- Ta.wiktionary.org [illegible] API [illegible]
- Ta.wiktionary.org [illegible]
- [illegible]

- [illegible]
- [illegible] FreeTamilEbooks.com [illegible] Google Play Books, GoodReads.com [illegible]
- [illegible]
- [illegible] (aamozish.com/Course_preface [illegible])

[illegible] kaniyamfoundation@gmail.com [illegible].

22.6 [illegible]

[illegible], 100% [illegible].

[illegible], GNU GPL, Apache, BSD, MIT, Mozilla [illegible].

22.7 [illegible]

[illegible].

[illegible] kaniyamfoundation@gmail.com [illegible].

Kaniyam Foundation
Account Number : 606 1010 100 502 79

Union Bank Of India
West Tambaram, Chennai

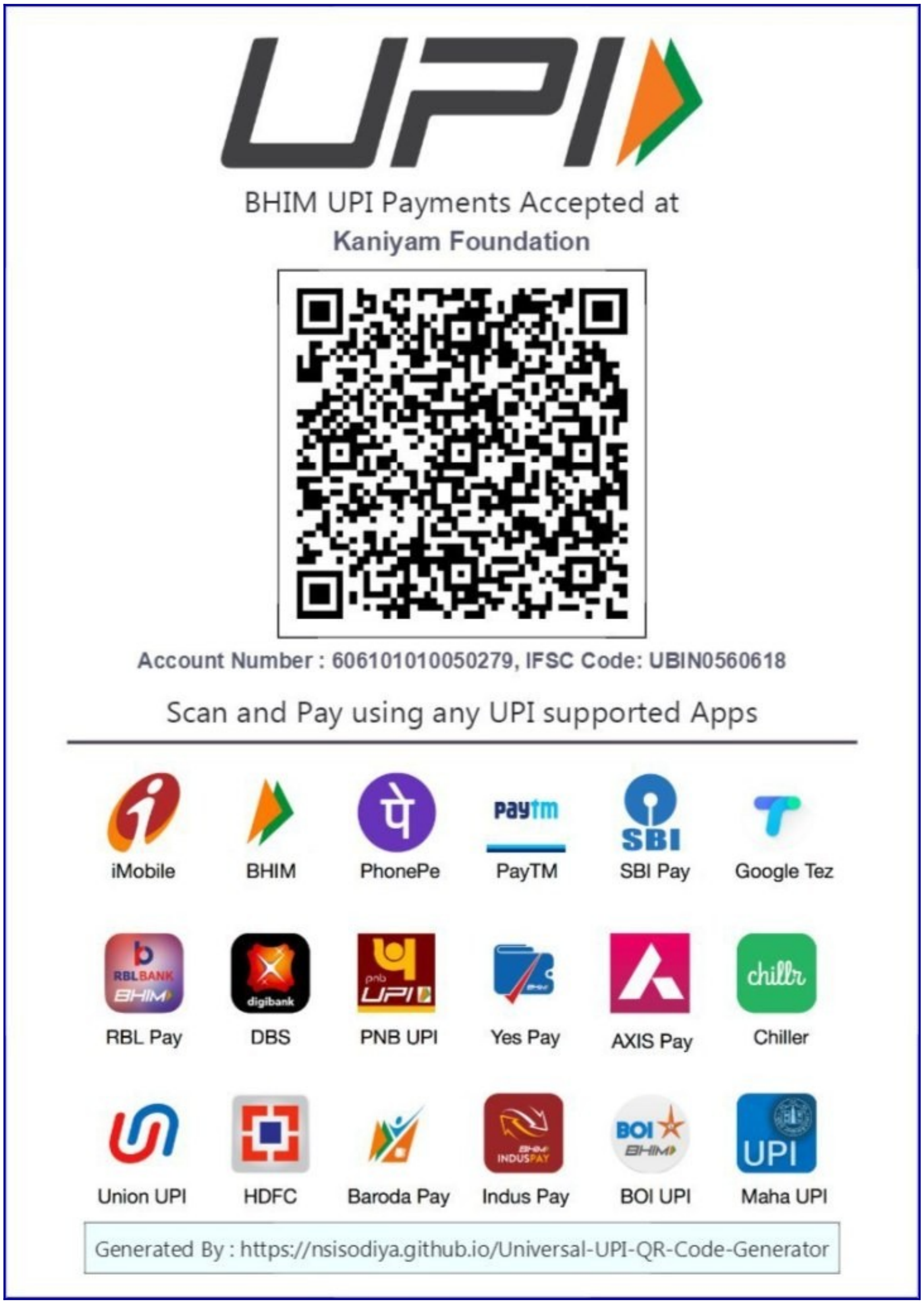
IFSC – UBIN0560618

Account Type : Current Account

22.8 UPI [illegible] QR Code

[illegible]: [illegible] UPI [illegible] QR Code [illegible].
[illegible], IFSC code [illegible].

Note: Sometimes UPI does not work properly, in that case kindly use Account number and IFSC code for internet banking.


UPI
BHIM UPI Payments Accepted at
Kaniyam Foundation
Account Number : 606101010050279, IFSC Code: UBIN0560618
Scan and Pay using any UPI supported Apps
iMobile
BHIM
PhonePe
PayTM
SBI Pay
Google Tez
RBL Pay
DBS
PNB UPI
Yes Pay
AXIS Pay
Chiller
Union UPI
HDFC
Baroda Pay
Indus Pay
BOI UPI
Maha UPI
Generated By : https://nsisodiya.github.io/Universal-UPI-QR-Code-Generator